DYNAMIC ROUTING AND RATE CONTROL IN STOCHASTIC NETWORK OPTIMIZATION: FROM THEORY TO PRACTICE

by

Scott Moeller

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

December 2010
Acknowledgements

The work presented in this document is the result of many collaborations and discussions. Chapter 3 was published in “IEEE Information Processing in Sensor Networks” (IPSN) 2010, and was the product of a collaboration between myself, Avinash Sridharan, Professor Bhaskar Krishnamachari, and Omprakash Gnawali. Chapter 4 was assisted by several very useful discussions with Professor Michael Neely and Longbo Huang. All work was substantially guided by my advisor, Bhaskar.

In 2008, Bhaskar received the USC Mellon Award for excellence in mentoring. This was no fluke, he takes a very active role in caring for and molding his students while shepherding them through the Ph.D. process. Over the course of my four years under his guidance, life has delivered some substantial bumps (to myself and others in the research group). Bhaskar has been more than a mentor, he has been a very valuable friend.

I would also like to acknowledge the support of Kathryn Brennan. There were certainly periods of substantial stresses caused by deadlines and examinations. Katie was supportive and comforting, and gave me the firm foundation upon which I could cling to in believing everything would work out.

I would like to thank Mike Neely for his numerous discussions with me, and Ramesh Govindan and his group for assistance in access to the Tutornet testbed. Specifically,
I would like to thank Luis D. Pedrosa, Nilesh Mishra, Marcos Vieira, Joongheon Kim, Jeongyeup Paek, and Ki-Young Jang for their frequent and timely assistance in Testbed maintenance. I relied heavily upon their support to complete the Tutornet experiments.

Finally, I would like to thank my parents and stepmother. Their love and the life experiences they made sure I collected have molded who I am today.
Table of Contents

Acknowledgements ii
List Of Tables vii
List Of Figures viii
Abstract xi

Chapter 1: Introduction 1
1.1 Motivation and Approach ........................................... 1
1.2 Contributions and Organization .................................... 6

Chapter 2: State of the Art 8
2.1 Wireless Sensor Network Applications .............................. 8
2.2 Wireless Channel Characteristics and Estimation ................ 13
  2.2.1 Hybrid Beacon and Data Driven Estimation ..................... 16
  2.2.2 Leveraging Short Term Quality Links ............................ 17
2.3 MAC Protocols and Radio Sleep Scheduling ..................... 19
2.4 Multi-Hop Wireless Routing ........................................ 20
  2.4.1 Quasi-Static Tree Routing ..................................... 20
  2.4.2 Dynamic Tree Routing ......................................... 23
  2.4.3 Tiered Routing ................................................ 26
  2.4.4 Any-to-Any Routing with Low State Space ..................... 28
  2.4.5 Routing Under Mobility ....................................... 31
  2.4.6 Cooperative Routing ......................................... 34
  2.4.7 Potential Routing ............................................ 40
  2.4.8 Backpressure Routing ......................................... 41
    2.4.8.1 Theoretical Work in Backpressure Stacks ................. 42
    2.4.8.2 Addressing Delay in Backpressure Systems ............... 46
    2.4.8.3 Network Scalability ...................................... 51
    2.4.8.4 Throughput Optimal Scheduling ......................... 53
2.5 Source Rate Control .............................................. 54
  2.5.1 Source Rate Fairness Objectives ............................... 55
  2.5.2 Source Rate Control for WSN .................................. 57
  2.5.3 Backpressure-Based Rate Control Mechanisms ................ 62

iv
Chapter 3: The Backpressure Collection Protocol

3.1 Introduction .............................................. 68
3.2 Backpressure Explained ................................. 71
  3.2.1 Routing as a Stochastic Optimization Problem ........ 72
  3.2.2 A Simple Example .................................. 74
3.3 Novel Contributions of BCP ............................ 75
  3.3.1 ETX Minimization ................................. 76
  3.3.2 Delay Reduction using LIFO ....................... 77
  3.3.3 Scalability ....................................... 79
3.4 BCP Implementation ................................... 80
  3.4.1 Routing and Forwarding ........................... 81
  3.4.2 Weight Recalculation .............................. 81
  3.4.3 Link Metric Estimation ............................ 81
  3.4.4 Disseminating Local Queue Backlog ................ 82
  3.4.5 Floating Queue Implementation ................... 83
3.5 Experimental Results .................................. 83
  3.5.1 Experimental Methodology ......................... 83
  3.5.2 Static Network Tests .............................. 84
    3.5.2.1 Delay Performance ......................... 85
    3.5.2.2 Scalability .................................. 88
    3.5.2.3 Goodput and Delivery Efficiency ............ 91
  3.5.3 External Interference .............................. 93
  3.5.4 Highly Mobile Sinks ............................. 95
  3.5.5 Application Experiment ............................ 97
3.6 Related Work .......................................... 101
3.7 Conclusion and Future Work ......................... 103

Chapter 4: Floating LIFO Delay Performance and Parameter Evaluation

4.1 Theoretical Analysis of Floating LIFO Queues ........ 105
  4.1.1 Floating Queue Bound on Discard Rate ................ 105
    4.1.1.1 Lagrange Multiplier Network Gravity Results of Huang and Neely ....................... 106
    4.1.1.2 The Floating Queue Algorithm ................ 108
    4.1.1.3 Preliminaries ................................ 111
    4.1.1.4 Bounding Floating Queue Discard Rates ........ 115
  4.1.2 Analysis of the LIFO Delay Advantage ............. 120
4.2 Floating LIFO Parameter Validation .................. 122
  4.2.1 Testbed and General Setup ......................... 123
  4.2.2 Experiment Parameters ............................. 125
  4.2.3 Results ........................................ 125
## Chapter 5: Rate Control and Dynamic Routing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Selecting a Source Rate Utility Function</td>
<td>130</td>
</tr>
<tr>
<td>5.2 The Theory Behind Backpressure Source Rate Control</td>
<td>131</td>
</tr>
<tr>
<td>5.3 Implementation Details in BCP</td>
<td>133</td>
</tr>
<tr>
<td>5.4 Empirical Max-Min Fair Rate</td>
<td>135</td>
</tr>
<tr>
<td>5.5 Rate Control Experimental Setup</td>
<td>138</td>
</tr>
<tr>
<td>5.6 Alpha-Fair Approximates Max-Min Capacity</td>
<td>139</td>
</tr>
<tr>
<td>5.7 Tension Between $V_{ETX}$ and $V_{alpha}$ Parameters</td>
<td>143</td>
</tr>
<tr>
<td>5.7.1 Proportional Fair Experimental Results</td>
<td>143</td>
</tr>
<tr>
<td>5.8 Rate Controller Parameter Sensitivity</td>
<td>146</td>
</tr>
<tr>
<td>5.9 Comparison with WRCP</td>
<td>149</td>
</tr>
<tr>
<td>5.10 Summary and Discussion</td>
<td>150</td>
</tr>
</tbody>
</table>

## Chapter 6: Conclusions and Future Work

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Conclusions</td>
<td>153</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>155</td>
</tr>
<tr>
<td>6.2.1 Performance Under Node Mobility</td>
<td>155</td>
</tr>
<tr>
<td>6.2.2 Handling Traffic Dynamics</td>
<td>156</td>
</tr>
<tr>
<td>6.2.3 Receiver Diversity</td>
<td>157</td>
</tr>
<tr>
<td>6.2.4 Multichannel Operation</td>
<td>158</td>
</tr>
<tr>
<td>6.2.5 Radio Sleep Scheduling</td>
<td>160</td>
</tr>
<tr>
<td>6.2.6 Multicast and Any-to-Any Unicast Routing</td>
<td>161</td>
</tr>
<tr>
<td>6.2.7 Throughput Optimal MAC</td>
<td>163</td>
</tr>
<tr>
<td>6.2.8 Rate Control Parameter Adaptation</td>
<td>163</td>
</tr>
</tbody>
</table>

References                                                                 | 164   |

Appendix: **Proof of Delay Reduction Using LIFO Service Priority**      | 174   |
List Of Tables

3.1  Test results for highly mobile sink experiment at source rate 0.25 packets per second per source, provided alongside static network results from section 3.5.2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97

4.1  Definition of variables used in Floating Queue operation and proofs . . . . 109
List Of Figures

3.1 An intuitive example of backpressure routing on a four-node line network with FIFO queueing service. Three packets (in black) are injected at nodes 1 and 2 at time B, intended for the destination sink S. ................. 71

3.2 A three node network is given in (i), links are labeled with both rate and expected transmission count per packet. Bold links in (ii) through (iv) indicate links selected for packet forwarding. Weights are calculated using Equation (3.5) with $V = 1$. ...................... 75

3.3 The four-node network of Figure 3.1, now with LIFO service priority. New additions to the queues flow over the existing gradient to the sink. .... 77

3.4 Floating queues drop from the data queue during overflow, placing the discards within an underlying virtual queue. Services that cause data queue underflows generate null packets, reducing the virtual queue size. .... 79

3.5 Source to sink delay CDF at 0.25 PPS for motes 4 and 40 under CTP, BCP-FIFO and BCP-LIFO. ......................... 86

3.6 The Reordering Density for BCP under FIFO (top) and LIFO (bottom) servicing priorities for 0.25, 1.0 and 1.5 packets per second per source. The quasi-static tree routing mechanisms of CTP resulted in greater than 99.9% in-order delivery for 0.25 and 1.0 PPS tests. .................. 87

3.7 Comparison of BCP’s per mote goodput, time average queue sizes and source-to-sink average packet transmissions per packet per source (with 95% confidence interval). Tests are run with and without BCP’s floating queues disabled. The maximum data queue size is 11. ................. 89

3.8 Goodput versus source rate in static network tests of BCP and CTP. ... 91

3.9 Average source-to-sink transmission count per packet per source (with 95% confidence interval) for the static 1.0 PPS experiment. Flow sources are sorted by average transmission count for BCP. ...................... 92
3.10 30 second windowed average sourced packet delivery ratio (top) and system transmissions per packet (middle). Spectrum analyzer results are plotted at bottom for the interfering 802.11 channel 14 traffic. .......................... 94

3.11 Comparison of performance of BCP and CTP under extreme sink mobility. 96

3.12 The sample path taken by the walking student through the 4th floor of RTH. 99

3.13 Comparison of localization performance for a sample run. The rear-network originated losses to which CTP is prone (due to queue tail drops near the sink) are causing loss of RSSI measurements in the rear of the network, thereby hampering localization performance. .......................... 100

4.1 The sample path for a Floating Queue beginning operation. Cross-hatch bars represent the data queue, gray bars represent the virtual data queue. Null packets are generated when the data queue underflows. ............... 109

4.2 The 40 Tmote Sky devices used in experimentation on Tutornet. ....... 124

4.3 The floating LIFO queues drop from the data queue during overflow, placing the discards within an underlying virtual queue. Services that cause data queue underflows generate null packets, reducing the virtual queue size. ............... 124

4.4 System average source to sink packet delay for BCP FIFO versus BCP LIFO implementation over various V parameter settings. ......................... 126

4.5 System packet loss rate of BCP LIFO implementation over various V parameter settings. ................................................................. 126

4.6 Histogram of queue backlog frequency for rear-network-node 38 over various V settings. ................................................................. 128

5.1 Per-source goodput, average packet transmission count, and delivery percentage for $V_{ETX} = 2$ and $\lambda \in \{3.75, 4.1\}$ PPS/Source, poisson traffic. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals. Note that beyond 3.75 packets per second per source, system stability is compromised. ......... 137

5.2 Per-source goodput, average packet transmission count, and delivery percentage for both alpha-fair source rate utility with $V_{ETX} = 2$ and $V_{alpha} = 5.4$ and the empirically derived max-min fair experiment. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals. ......................... 140
5.3 Per-source goodput, average packet transmission count, and delivery percentage for both alpha-fair source rate utility with \( V_{ETX} = 2 \) and \( V_{\alpha} = 2.15 \) and the empirically derived max-min fair experiment. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.4 Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with \( V_{ETX} \in \{1, 2\} \) and \( V_{\alpha} = 2.5 \). As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.5 Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with \( V_{ETX} \in \{1, 2\} \) and \( V_{\alpha} = 1.88 \). As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.6 Per-source goodput, average packet transmission count, and delivery percentage for Proportional-fair source rate utility with \( V_{ETX} = 2 \) and \( V_{\text{prop}} = 20 \), \( V_{\alpha} = 5 \). As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.7 Per-source goodput, average packet transmission count, and delivery percentage for Proportional-fair source rate utility with \( V_{ETX} = 2 \) and \( V_{\text{prop}} = 5 \), \( V_{\alpha} = 2.15 \). As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.8 Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with \( V_{ETX} = 2 \) and \( V_{\alpha} \in \{2.5, 5\} \). As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.9 Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with \( \alpha = 4 \), \( V_{ETX} = 2 \) and \( V_{\alpha} = 2.5 \). Unlike prior sections, we enlarge our packet to 40 Bytes for proper comparison with the IFRC/WRCP results of [119]. Additionally, the sink node is 29. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.10 Per node routing churn for alpha-fair source rate utility with \( V_{ETX} = 2 \) and \( V_{\alpha} = 2.15 \).
Abstract

Real-world applications of wireless sensor networks are frequently faced with network capacity constraints, restricting the sensing frequency or scalability of the deployment. In the absence of transport-layer rate control the allocation of network capacity can be highly asymmetric, favoring sensing nodes near the collection agent. Further, external interference and new participatory sensing paradigms can result in highly dynamic collection topologies. Lastly, protocols for the resource-constrained networks must emphasize low complexity while minimizing control overhead. Addressing these challenges, we present a novel backpressure-based routing and rate-control stack that is motivated by stochastic network optimization theory.

Current data collection protocols for wireless sensor networks are mostly based on quasi-static minimum-cost routing trees. We first consider an alternative, highly-agile approach called backpressure routing, in which routing and forwarding decisions are made on a per-packet basis. Although there is considerable theoretical literature on backpressure routing, it has not been previously implemented on practical systems due to concerns of packet looping, the effect of link losses, large packet delays, and scalability. We present the Backpressure Collection Protocol (BCP) for sensor networks, the first-ever implementation of dynamic backpressure routing in wireless networks. In particular,
we demonstrate for the first time that replacing the traditional FIFO queue service in backpressure routing with LIFO queues reduces the average end-to-end packet delays for delivered packets drastically (75% under high load, 98% under low load). Further, we improve backpressure scalability by introducing a new concept of floating queues into the backpressure framework. Under static network settings, BCP shows a more than 60% improvement in max-min rate over the state of the art Collection Tree Protocol (CTP). We also empirically demonstrate the superior delivery performance of BCP in dynamic network settings, including conditions of extreme external interference and highly mobile sinks.

Backpressure-based stochastic network optimization theory employs a tunable optimization parameter, $V$. As $V$ is increased, utility or penalty performance approaches the optimal like $O(\frac{1}{V})$ while delay grows linearly in $V$. We provide analysis motivating the novel usage of the LIFO queueing discipline in backpressure stacks, suggesting that delay scales near-optimally like $O(\log^2(V))$ for all but a small fraction of traffic. We then empirically evaluate delay and discard performance of BCP as the $V$ parameter is raised, and find the results in strong agreement with theory.

Finally, we turn our attention to state of the art rate control protocols for wireless sensor networks, which are traditionally assumed to run atop the aforementioned quasi-static routing trees. We implement and empirically explore a backpressure rate controller, theoretically capable of maximizing an aggregate source utility function while the underlying backpressure routing framework dynamically routes packets, often amongst multiple paths, to the collection agent. We demonstrate an alpha-fair rate controller which
achieves 95% of the empirically determined max-min fair rate allocation over a 20-mote deployment, and 80% of the max-min fair rate allocation over 40 motes.
Chapter 1

Introduction

1.1 Motivation and Approach

In the last two decades, a new domain of wireless deployments has emerged as a highly capable solution to a number of real world challenges in sensing. Specifically, there exist applications for which sensory data is desired from a large number of sample points, for extended periods, at low cost and with minimal impact or effort in deployment. A good example can be found in the monitoring of Leach’s Storm Petrels on Great Duck Island in Maine [84]. Human presence not only impacts the behavior of the birds and damages their habitat, but biologists have discovered that as little as 15 minutes of human activity in a bird colony can lead to 20% mortality among eggs and chicks in a breeding year [6].

Given this extreme sensitivity, how can biologists monitor these bird populations without doing harm? The solution arrived at in [84] was to place Wireless Sensor Network (WSN) devices in the nesting locations in early spring, before the nesting birds even arrived. The WSN devices supported wireless collection of sensor data, and were relatively inexpensive, flexible, and capable of robust deployment durations even with limited
battery resources. This allowed the researchers to monitor nesting behavior through the entire nesting season without visiting the nest site.

This is just one example of the specialty domain in which WSN are much better suited than any alternative developed technology. Other published deployments include habitat monitoring [84, 72], Cane Toad monitoring [48], redwood tree micro-climate survey [130], industrial monitoring [72], structural monitoring [146, 103, 125], traffic monitoring [11], bridge monitoring [68, 22], volcano monitoring [141, 142, 140], forest fire risk monitoring [42], vehicle tracking [43], animal tracking [151, 129, 138], bull electronic fences [138], and noise pollution monitoring [30].

Though these WSN devices are well suited to this broad range of sensing applications, the devices pose a broad set of challenges due to the hardware design space. One common platform is the Tmote Sky device. Packed in a footprint approximately the size of two AA batteries, the Tmote Sky carries a Chipcon CC2420 radio, a TI MSP430 microprocessor, 10kB RAM and 48kB of Flash memory. The CC2420 radio is 802.15.4 complaint, consumes approximately 18mA at 3V when receiving or transmitting, and has a maximum transfer rate of 250kbps (although realistic single-hop rates rarely exceed half this rate). The radio can be placed in sleep mode to varying degrees, trading listening capability and activation time for power consumption. The MSP430 is a 16-bit microprocessor clocked at 8Mhz and is also capable of being put to sleep (where it consumes micro-amps). By comparison, we note that the latest TI-89 graphing calculators operate at 16Mhz on 32-bit register files while supplying 188kB of RAM and 369kB of FLASH memory. The WSN hardware therefore has only a fraction of the resources provided by a modern graphing calculator.
The extremely limited resources, coupled with substantial challenges posed by low power radio transmissions, has necessitated a bottom-up exploration of multi-hop routing for the purpose of data collection in WSN. Early research in link characterization and routing performance for data collection [145, 146, 103] established key design decisions upon which subsequent data collection protocols for WSN were built. As a general rule, collection protocols have emphasized the detection of long-term reliable links and the construction of static or quasi-static routing trees (e.g. Wisden [103], MintRoute [145], MultihopLQI, the Collection Tree Protocol [38]). This design decision is rooted in two primary concerns: looping behavior during transients in routing tree construction and control message overhead in tree re-construction upon routing modification. The advantage of such approaches is a very high packet delivery reliability and in-order packet delivery.

In modern collection protocols, such as the Collection Tree Protocol (CTP [38]), delivery ratios for low source rates and moderate size topologies are routinely in the high 99% range. Further, in this rate regime, the tree re-generation rate has been shown to cause low overhead thanks to exponential beaconing back-off mechanisms employed. Not all applications are capable of operating under such low data rates however, and to date there has been little or no exploration of multi-hop routing in the face of high topology dynamics. Many of the real-world deployments cited above have stressed the difficulties encountered due to their high data rate sampling requirements. One may argue that these applications are simply incompatible with the constraints faced by WSN hardware, but we note that even very low sampling rate applications experience radio rate collapse.
when faced with a network of very large scale. We contend that enhancing the capacity region of WSN is of prime importance.

Additionally, new sensing paradigms (e.g., participatory sensing [20]) may result in data collection to a mobile agent opportunistically passing through the sensing field. Several multi-hop routing algorithms have been derived for this domain (SPEED[44], Hyper[117]), and though not designed for mobility, the authors of CTP found their TinyOS 2.x protocol out-performed Hyper [38] after porting the protocol from TinyOS 1.x. While Hyper and CTP rely on tree construction, SPEED does not maintain routing state. This design decision was made by authors of SPEED because they recognized the stability concerns and overhead associated with collection tree generation in highly dynamic topologies. The solution used in SPEED is that of greedy geographic routing in combination with delay estimation and a push-back technique to divert traffic around congestion and voids. Being a geographic routing protocol, nodes must know their location and void avoidance results in non-optimal routing stretch (that is, packets are transmitted further than the shortest path).

In this thesis, we aim to demonstrate for the first time a nearly complete backpressure network stack, including dynamic routing and rate control components. In doing so, we first target settings in which WSN face source loading that threatens network stability under traditional tree routing mechanisms and second, emphasize the performance of these network stacks in the presence of extreme dynamics (such as posed by participatory sensing, for example).

Conceptually, a fluid flow model approximates the backpressure stack with reasonable fidelity. At each source, water pours from a faucet into a basin and as water backs up it
begins to flow toward the basin exit (or sink). The faster the sources admit water, the
more volume that remains standing in the basin at any one time. The water level at any
given source (or faucet) would then represent the degree of congestion. Referring back to
our WSN routing and rate control objectives, each node computes a per-neighbor weight
that indicates the desirability of forwarding a packet in that direction. Nodes forward
packets when weights are positive, therefore forming a packet backlog gradient toward
the sink. As admissions progress and queues stabilize, the local queue backlogs can be
used to indicate both routing cost to the sink and congestion in the network. Theory
indicates that the local queue knowledge can be used to implement source rate controllers,
necessary to avoid network congestive collapse and maintain fairness amongst the many
sources in the network.

A backpressure stack therefore computes per-packet and per-hop routing decisions,
potentially making the protocol very robust to dynamics. Further, a backpressure stack
that supports routing is inherently multi-path and avoids queue tail drops, enabling
greater network collection capacity.

In the theoretical utility optimizing backpressure frameworks, upon which our systems
implementation is based, a constant parameter tunes the relative importance of queue
size and the utility (or penalty) optimization. This parameter is often denoted as $V$, and
as it is increased it has been shown that time average queue size grows linearly while the
asymptotic system performance with respect to utility or penalty approach the optimal
value inversely with $V$. 
1.2 Contributions and Organization

The contribution of this thesis is in the implementation and preliminary evaluation of backpressure routing and rate control, derived from recent theoretical work in stochastic network optimization. This is the first such implementation and empirical evaluation in a wireless sensor network testbed. As there does not yet exist a systems-tested theoretically throughput optimal MAC for wireless sensor networks, our stack runs atop the default CSMA MAC for which no throughput guarantees are known. As a result of this challenge, prior to the work presented in this document, it was unknown whether backpressure stacks could even successfully operate without first addressing this MAC layer deficiency.

In implementing the Backpressure Collection Protocol (BCP), a dynamic backpressure routing collection protocol for WSN, we discovered a number of theory-to-practice barriers. We describe in Chapter 3 our work in creating BCP, in which we encountered several key difficulties: packet looping and subsequent losses due to link layer failures, backpressure failure due to finite queue overflows, and severe packet delivery latencies at low source rates. We addressed each of these challenges to a degree sufficient to prove backpressure routing competitive with existing tree routing techniques, even in static settings. To remove packet looping we used Expected Transmission Count (ETX) link penalties within the stochastic network optimization framework. Addressing finite queue overflows, we leveraged recent theoretical queue gravity results and implemented floating data queues. And through novel application of LIFO queueing priority, we demonstrated a reduction greater than 98% in packet delivery latency for low source rates.
Our work in Chapter 3 motivated our subsequent theoretical analysis of the LIFO delay advantage, which we explore in Chapter 4. We prove that for data queue sizing which is $O(\log^2(V))$, our floating queue achieves a packet discard rate that scales like $O\left(1/V^{\log(V)}\right)$ for networks satisfying the assumptions of the queue gravity work presented in Huang and Neely [50]. We note that recent work by Huang et al. has substantially strengthened this theoretical result for LIFO queues in backpressure frameworks [49]. We then provide empirical testbed validation of this scaling property using our BCP deployment.

A third contribution is detailed in Chapter 5, where we move up one layer to implement source rate utility maximization atop our BCP implementation. We explore for the first time the performance of backpressure rate control atop a backpressure dynamic routing stack (here BCP), and the real system impacts of multi-objective optimization within the stochastic network optimization framework. The optimization goals provided here are in tension; we aim to both minimize packet transmissions (ETX) and maximize aggregate source utility. We show empirically that for suitable selection of the optimization parameters we can approximate max-min fair source rate allocations reasonably well. Through empirical evaluations of proportional fair objectives we also demonstrate the flexibility of the backpressure stack to accept alternative source rate utility functions.

Finally, we conclude by providing numerous areas for future research in Chapter 6.
Chapter 2

State of the Art

The body of related work applicable to backpressure routing is very broad. First in this section, we’ll discuss related work on WSN protocols that influenced BCP design. Our system’s translation of backpressure theory is informed by previous work in MAC layer research, realistic link stochastics and estimation techniques, and our own prior back-pressure work in source utility optimization over fixed routing topologies. Second, we’ll highlight related theoretical work. Our efforts in this domain were motivated by a string of recent publications on stochastic network optimization, which has been rapidly evolving. We will also discuss extensions of Lyapunov optimization techniques, not included in BCP, and the potential future systems work that could result. Finally, we’ll discuss the related systems research with which BCP competes for adoption. We’ll highlight the strengths and weaknesses of BCP when compared with alternative system developments.

2.1 Wireless Sensor Network Applications

There are many real-world wireless sensor network studies, including Habitat monitoring [84, 72], Cane Toad monitoring [48], bird nest monitoring [84], redwood tree micro-climate
survey [130], industrial monitoring [72], structural monitoring [146, 103, 125], traffic monitoring [11], bridge monitoring [68, 22], volcano monitoring [141, 142, 140], forest fire risk monitoring [42], vehicle tracking [43], animal tracking [151, 129, 138], bull electronic fences [138], and noise pollution monitoring [30].

Prior work in some domains has clearly demonstrated the need for greater network capacity. Examples include cane toad monitoring [48], water pipe integrity monitoring [125], volcano monitoring [142, 140], bridge monitoring [68], and structural health monitoring [103]. A number of solutions to this problem are implemented in practice, including in-mote FFT processing, event detection or other compression (early volcano monitoring [142], [48],[125], Wisden [103]), feature processing and dissemination for selective recall (updated volcano monitoring via Lance [140]), and application-specific spatial reuse (Golden Gate Bridge [68]).

In one of the earliest system demonstrations of high sampling rate real-world deployments, Wisden [103] handles high frequency data sampling by run length compression coupled with event onset detection. During times of perceived disinterest, compressed samples were forwarded to the collection sink. The onset detector maintains noise mean, noise standard deviation, and signal envelope; it therefore consumes low resource overhead as no sample storage is required for operation. In empirical tests over 14 MicahZ motes in an earthquake test structure, a sampling rate of 200 Hz (single axis) has been achieved. The authors note that in their application, the deployment has ample time to collect raw accelerometer data after the event occurs (they provided five minutes for
complete collection). This is necessary, as the 200 Hz sampling resulted in 11.1 packets/sec during non-quiescent periods, while the multi-hop routing network as configured supports only 2 packets/second.

In an early work by Werner-Allen et al., the authors aim for reliable retrieval from a high sampling frequency volcano monitoring sensor network over a their FETCH protocol [141]. The authors recognize that efficient transfers are possible when the multi-hop network access is Time Devision Mulitplexed (TDM’d) among sources (see traffic monitoring [11], golden gate deployment [68] for similar findings). FETCH is therefore designed to detect a volcanic event, wait for complete data collection (30 seconds) by sensors, halt sampling by all nodes in the network and then one node at a time request data blocks until sixty seconds of measurements had been collected from every node. These protocol decisions are driven by the high sampling rate and low storage capability of the sensor devices, and by the need for reliable data collection. Subsequent work by Werner-Allen et al. considers the energy and network capacity restrictions in data collection for high frequency sensor networks [140]. Their LANCE architecture optimizes the utility of collected data, subject to energy considerations and network capacity. The architecture handles an over-abundance of sampled data by operating in a centralized paradigm, in which only summary features of the sensed data are transferred over a spanning tree to a controller. The controller, knowledgable of the network topology and therefore energy consumption in retrieving from individual nodes, computes a cost-benefit optimization and uses Fetch [141] reliable transport protocol to collect the requested data units for select sensed events.
Even applying these data reduction techniques, the authors of these works note the advantage that would be found in a greater data collection capacity. In the early volcano monitoring work [142] it was found that reliable collection of 60 seconds worth of seismic data from 16 nodes took roughly one hour to accomplish. This resulted in many lost events when compared with a simultaneously deployed high fidelity solid state storage seismometer.

In the work by Kim et al. [68], the authors are able to stream 441B/s from the 46th hop of their Golden Gate monitoring wireless sensor network. In order to accomplish this data rate at such deep hop count, they implement a number of application specific techniques including spatial pipelining of transfers along the bridge. Like Lance, the authors implement a reliable fetch protocol, here called Straw. The routing tree is generated by MintRoute [145], and inter-packet transfer periods are set to the hop count to the sink or five transmission periods. This introduces the pipeline effect which leverages spatial diversity along the bridge. The authors also note that small packet sizes are introducing high header overhead. They conclude that even though larger packets result in higher loss percentage due to collisions, some moderate degree of data aggregation and block transfer is most efficient.

In combatting congestion, explicit scheduling of data transfers is used by both [48] (which duty-cycles paired acoustic sensors in order to maintain listening activity and reduce network transfers) and [11] (which explicitly schedules in a TDMA fashion all multi-hop collection flows). The technique employed in [11] is similar to Flush [67], but simplified for the application at hand (being only four to six hops in depth). Specifically, the authors pipeline transfers along the line of car-sensing motes such that each transfer
occurs spatially while the vehicle is passing, allowing aggregation and spatial separation
in the pipelined collection along the motes. In another setting, the bull virtual fence work
of [138] uses not only a one-hop diameter network, but is also able to schedule the MAC
to have low collision probability (i.e. a TDMA MAC). For the application at hand, the
loss rate due to collisions was acceptable and no MAC retransmissions were attempted.

Some use a reliable transport protocol (e.g. Intel Mote implementation in PipeNet
[125], Fetch implementation for Lance [141], and Straw for the Golden Gate bridge de-
deployment [68]), others accept packet loss rates (e.g. steer animal control [138]). Reli-
able delivery on the motes is not entirely sufficient, however. The PipeNet deployment
demonstrates long term sample delivery ratios (during operational days, omitting battery
outages) of 87% and 62% for two of the deployed sensing clusters. Losses were frequently
due to battery outage or packet loss over GPRS or cellular backhaul outages. Single tier
networks with end-to-end reliable delivery or retrieval (e.g. Wisden [103]) demonstrate
100% data delivery rates. This may highlight the need to consider the entire system of
systems in reliable delivery.

It is worth noting that not all applications require reliable delivery of sample data.
Results obtained from the Redwood macro-scope study [130] found a data sample yield
of only 49%. Yet substantial biological findings were still possible with proper analytical
techniques. The authors further note that even successfully collected sensor samples are
not guaranteed to be accurate. Outlier detection is required in order to filter out faulty
data that had been corrupted either by sensor malfunction, unexpected and brief direct
sunlight exposure, or in network transport (which would require sophisticated checksums
or forward error correction to combat). Therefore, aiming for 100% delivery is not the only consideration in comprehensive sample collection.

While Energy budgeting and rationing is often a concern in Wireless Sensor Networks, the battery life in PipeNet (6V 12Ah battery, [125]) is consistently found to be around 50-62 days. The Golden Gate deployment survived 23 days on an 18Ah battery [68]. Both works discuss the advantages of longer up-time and issues with battery maintenance for real-world adoption, but highlight for us the fact that these settings cannot simultaneously support high sampling frequency and long lifetime. In other applications with high sampling frequency, there is often a lesser necessity for battery longevity. Clearly we would prefer both high collection capacity and long battery life, but the two objectives are under tension. In exploring backpressure routing, we focus in this work primarily on enhancing the capacity of the network and robustness to mobility and external interference events.

2.2 Wireless Channel Characteristics and Estimation

In a ground-breaking paper that questions prior assumptions, De Couto et al. investigates poor performance on their wireless ad-hoc network in their laboratory at M.I.T. [26]. The authors deploy DSDV [105] over both an 18-node indoor network as well as the outdoors roofnet, noting that for static networks, the performance of DSDV, DSR and AODV has been demonstrated to be comparable [17]. In their empirical experimentation, De Couto et al. discover that contrary to common belief (often motivated by simulation work), the
use of hop count as a routing metric in these static ad-hoc settings frequently results in poor network performance.

In exploring this performance issue, they found that though there were a wide variety of link reliabilities in the deployments, there existed paths with greater than 90% packet delivery in all cases. It is therefore possible to have reliable routes, but these routes are not always selected by DSDV. Further investigation by the authors yields an interesting discovery. By comparison with maximum throughput routes that have been hand filtered, De Couto et al. discovered that the maximum throughput routes were indeed routes having minimum hop count, as noted in [79]. But there are typically a multiplicity of alternative routes having this minimum hop count, with wide variance in throughput. In effect, the blacklisting mechanisms of DSDV was removing the worst links, but then allowing minimum hop count routes to be formed from the soup of sometimes sup-optimal link alternatives. Specifically, they discover that among the alternative min-hop paths, a randomly selected route would achieve less than half the maximum throughput more than half the time. This discovery then leads the authors to investigate more thoroughly the characterization of link reliability and its inclusion in routing metrics.

These efforts result in the introduction of Expected Transmission Count (ETX) as a new routing metric, published by De Couto et al. in 2003 [25]. In careful examination of the impacts of a minimum-hop routing metric within DSDV or DSR, they expose the weaknesses of such an approach within the wireless setting. Specifically, longer hops tend to have intermediate reception probabilities, a feature unplanned for by a pure hop-minimization approach. Further, in agreement with [26], the authors note that even when the hop-minimization could lead to a high throughput path (e.g. dense networks),
there are often multiple alternatives that minimize hop count and have widely varying throughput capability.

The ETX metric of De Couto et al. [25] has been incorporated into DSDV and DSR routing protocols. All nodes beacon every $\tau$ seconds, with a small randomized jitter to prevent collisions. By observing the number of beacons overheard in a ten second window, each node computes per-link unidirectional delivery ratios (say $d_f$ is the forward link delivery ratio for a particular link). By inserting these windowed $d_f$ estimates in the beacons as they are broadcast, all neighbors learn the $d_r$ reverse delivery probabilities per link. The link ETX is then $\frac{1}{d_f \cdot d_r}$. Handily, the path ETX is simply the sum of the constituent link ETX values, allowing for common shortest-path algorithms to be run without application of natural logs or other tricks.

De Couto et al. then tested experimentally on a 29-node 802.11b office deployment. They find that the shift from hop-minimization to ETX-minimization yields throughput up to two times higher, with indications that the gains would scale even further as network diameter increased. The authors note that under un-modified DSDV or DSR, the network is nearly unusable - the throughput performance was simply unacceptable. Further, as a side benefit, minimizing ETX may also reduces energy consumed per packet delivered thanks to reductions in radio usage.

In the first medium to large size link characterization study of wireless sensor networks, Zhao and Govindan [152] note the generally poor packet reception probability of these links. On Mica motes with 433 Mhz radios and custom omni-directional antennas, they demonstrate that for a large percentage of links the reliability is highly temporal in nature. Even for links that may appear reliable, they find that in some test environments fully
10% of links are asymmetric in nature. This is quite problematic for common Automatic Repeat Request MAC mechanisms implemented in these light-weight systems.

Interestingly, the results of [152] indicates better packet reception characteristics for lower transmit power. The authors hypothesize that the shorter physical range allows for lesser multi-path opportunities. We note that this motivates multi-hop deployments with shorter links and lower transmit powers. The “gray” distance in which packet reception is temporal and sometimes very poor proves to be quite wide. Zhao and Govindan hypothesize that in these radio networks, nodes may need to carefully select neighbors based upon measured packet delivery performance, not based upon short-term signal strength metrics. Specifically, Zhao and Govindan find that while reliable links had high receive signal strength, no all links with high receive signal strength have low packet loss. Therefore, receive signal strength is not always a reliable indicator of link quality.

Subsequent investigation into performance of 802.11b by Aguayo et al. yields similar findings [2]. They collect link statistics for the 38 node 802.11b Roofnet in Cambridge. Among the similar results is the large number of poor reliability neighbor links, and the burstiness of packet failures on these collections of intermediate quality links. The authors hint at the self-similarity of the link quality, indicating that a short term estimate is accurate for short term prediction, but that over longer durations the short term estimate is not useful.

### 2.2.1 Hybrid Beacon and Data Driven Estimation

The findings of Zhao and Govindan motivate work by Fonseca et al. into a new link estimator for multi-hop routing [31]. Extending substantially the work of Woo et al.
Fonseca et al. propose and validate the substantial gains of a distilled cross-layer link estimator dubbed the four-bit link estimator. Two primary contributions arise from their work. First, they demonstrate the challenges faced by state-of-the-art link estimators when working with realistic neighbor table space. Fonseca et al. show empirically the poor routing selection choices that can result from basic implementation of neighbor table management. Second, the authors demonstrate that with only minimal information from each of the physical, link, and network layers (four bits, to be precise), great gains in neighbor table entry selection and link estimation can be achieved. The authors clearly rationalize their metrics in the context of the link challenges discussed in [152, 123], namely the high number of neighbors with poor link quality and the highly temporal nature of link reliability. The four-bit link estimator implements a hybrid data-driven and beacon based technique, such that in times of high data traffic the ack mechanisms of the link layer dominate the link estimate, while at low traffic times the beacon receive signal strengths dominate the link quality estimate. Routing layer knowledge selects for which links will be monitored, therefore supporting a neighbor table with only ten entries while preserving multi-hop routing performance. In computing the data-driven link estimate, the four-bit link estimator makes use of the windowed exponential weighted moving average which performed well in Woo et al. [145].

2.2.2 Leveraging Short Term Quality Links

Srinivasan et al. characterize the transient link reliability of both 802.15.4 and 802.111 links. By looking at traces from three medium to large scale sensor network deployments, Srinivasan et al. describe the computation of a conditional packet reception cumulative
probability distribution function (CPDF) [124]. The authors demonstrate that a simple opportunistic transmission algorithm which transmits successive packets immediately until failure, then implements a fixed backoff interval, substantially improves the efficiency in both single and end-to-end transmissions per packet. The authors insert the opportunistic transmission algorithm into the Collection Tree Protocol and run experiments over 80 nodes in the Intel Mirage testbed, finding a 15% reduction in packet transmissions when a fixed backoff interval of 500 ms is employed. This efficiency gain comes at the cost of delivery delay. The authors point out that the worst case packet delivery latency increases between 4 and 25-fold depending upon transmit power (and therefore network diameter).

The self-similarity in link behavior, noted in [124], has recently been leveraged by systems researchers. Alizai et al. [5], using overheard packet transmissions, detect links that are temporarily reliable. Through burst usage of longer (metric) distance transmissions during periods of link reliability, the authors reduce per-packet average transmissions by 20% over the Collection Tree Protocol, and in doing so also raise the maximum sustainable throughput of CTP by 20%. Alizai et al. term their modification to CTP a Short Term Link Estimator (STLE). It is interesting to note that philosophically, the STLE mechanism is leveraging link temporal variability in exactly the opposite manner as proposed by Srinivasan et al. [124]. Instead of using packet failure to back off during rapid link transmissions (as was done in [124]), STLE uses consecutive successful receptions to trigger the use of a questionable but long-distance link.
2.3 MAC Protocols and Radio Sleep Scheduling

Within the context of real-world WSN deployments, the application space can be divided into high frequency sampling applications which require heavy data collection [68, 48], and applications requiring energy-efficient radio sleep duty cycling in order to support the desired deployment durations [72, 142]. There are two primary approaches employed by low power MAC that use radio duty cycling, each of which is important for certain applications. Protocols like Koala [89], leverage synchronous sleep cycling, wake all nodes through Low Power Probing (LPP) techniques. For delay-insensitive collection, synchronous sleep scheduling has been demonstrated with sleep duty cycles of 0.1% or lower, supporting low data rate collection at aggregate duty cycles of 0.2% or lower [89]. The long aggregation and collection cycles of synchronous sleep macs result in very high collection latency, which may be unacceptable under certain applications.

To satisfy lower delivery delay, asynchronous MAC techniques have been developed. One such MAC for TinyOS is the Low Power Listener implementation (a naive variant of WiseMAC [28] or B-MAC [108]). Under asynchronous MAC, nodes wake briefly in order to listen for packets being repeatedly broadcast by senders. Upon overhearing a transmitter, the listener checks the header to determine whether the device is the destination. If not, the MAC puts the radio back to sleep until the next listener cycle begins. In TinyOS 2.x, LPL has been found to support comparable sleep dying cycles to LPP (0.2% or lower [28]) but overhearing of regional packet transmissions results in higher aggregate duty cycles. In real world deployments, the Collection Tree Protocol for
TinyOS has been demonstrated to achieve radio duty cycles as low as 3% while supporting aggregate traffic rates of 30 packets/minute [38].

2.4 Multi-Hop Wireless Routing

2.4.1 Quasi-Static Tree Routing

The previously cited work by Woo et al. [145] drew a number of conclusions widely cited in systems design for tree routing algorithms in wireless sensor networks. The authors evaluated a broad spectrum of routing related issues in wireless sensor networks, and came to a number of cornerstone conclusions upon which many future tree routing algorithms have been based.

Noting that packet snooping can be achieved for free, and citing their own previous work on low power MAC which support snooping [144], Woo et al. leverage data-path packet snooping to detect looping behavior and gain awareness of neighboring children nodes within the tree structure. Even with these snooping mechanisms to reduce loop occurrences, some loops will occur. Woo et al. argue that a light-weight loop detection algorithm is best suited for wireless sensor networks, and upon detection they simply drop packets until the routing loop is removed.

Packet duplication is another source of difficulty for real world collection protocols in wireless sensor networks. The light-weight ack mechanisms, flakey link reliability and asymmetric links of 802.15.4 can frequently result in duplicate packet generation due to lost acks. In [144], Woo et al. maintain a 2-tuple within the neighbor table, in which
they store the most recent sender ID and packet sequence number received from that neighbor. Using this lightweight filter, duplicate packet receptions are easily filtered.

Concerned that nodes near the sink may find themselves unable to admit local traffic due to forwarding queue congestion, Woo et al. maintain separate forwarding and originating message queues and schedule between the two for transmissions. Strict priority is given to locally sourced messages, under the assumption that the ratio of locally sourced to forwarded data is very low.

Citing the less reliable links of 802.15.4 networks, when compared with their high power brethren, Woo et al. evaluate several alternative routing metrics. Included in their evaluation are min-hop, min-transmission (ETX), and path reliability (the product of link reliabilities in the path to the sink). Their results conclude that min-transmission with hysteresis is the routing metric of choice.

The link estimation and routing behavior conclusions arrived to in [145] were adopted first into the BLAST software prototype. This framework was then leveraged by Paek et al. in their application-tested collection stack called Wisden [103]. Their work on Wisden represents one of the earliest demonstrations of link estimation and multi-hop routing in wireless sensor networks. In [103], the authors note that only 41.3% of packets are delivered via one-hop routes, a result primarily of inter-node interference reducing the successful radio range of the MicaZ devices.

Following the early success of BLAST and Wisden, the community eventually evolved BLAST into the MintRoute collection protocol for TinyOS. Using link-layer acknowledgements, MintRoute maintains the min-transmission routing metric and therefore operates on packet Expected Transmission Count (ETX) as the metric for tree generation.
MintRoute is used in a number of the real-world applications discussed earlier, including the redwood micr-climate survey [130] and the Golden Gate Bridge work [68]. Further evolution of MintRoute lead to MultiHopLQI (available in TinyOS). At the time of its release it became the best performing collection protocol for TinyOS [31]. The MultiHopLQI collection protocol is used in the volcano monitoring deployment of Werner-Allen et al. [142].

While protocols discussed thus far have emphasized performance and reliability, there is little if any discussion of debug capability in the real world. The real world deployments discussed previously all struggled substantially in determining the modes of failure. Noting this, Wachs et al. propose a new metric for protocol termed Visibility, meaning the ease with which protocols can be debugged. Using wired backhaul logging of packet collection of MultiHopLQI on 31 devices within Motelab at Harvard [143], the authors parse the cause of all but 1% of packet losses in a number of collection scenarios. This allows a probability distribution of packet loss causes to be generated, over which the authors optimize a decision tree with debug energy costs. This is the Visibility metric.

After evaluating the expected cost of debug per lost packet within MultiHopLQI, the authors propose the first collection protocol specifically optimizing the Visibility metric. Wachs et al. describe five key design decisions for the Pull Collection Protocol (PCP) in [135], several of which have been maintained in our own implementation contribution. First, in order to minimize ingress drops (a result of packet arrival to a full queue), a pull architecture is implemented in PCP. Egress drops occur when the maximum retransmit attempts is met and the head-of-line packet is discarded. The authors therefore remove
egress drops through removal of this mechanism, which potentially requires mild additional stack complexity. *Node reboot* events are made easier to detect through a sequence number reset mechanism in PCP. *Node disconnection or death* is difficult to detect, the authors suggest that simply detecting no traffic from a node may suffice. Finally, *duplicate packet suppression* is a source of packet losses when temporary loops are generated. This is an artifact of the simple 2-tuple filter mechanisms of simple collection protocols. The authors implement a new 3-tuple, including hop count in the packet header, to overcome the discards associated with transient loops under the 2-tuple filters. When all features were combined into the Pull Collection Protocol, the authors demonstrated that the delivery percentages were much higher than IFRC [113] or MultiHopLQI [145].

### 2.4.2 Dynamic Tree Routing

In [105], Perkins and Bhagwat propose the Destination-Sequenced Distance-Vector routing protocol (DSDV). Their work is one of the first to look at the challenges of metric computation (here shortest path) in highly dynamic ad-hoc wireless networks. In their context, highly dynamic does not necessarily mean mobile, but implies a frequency of node join and departure that far exceeds the assumptions of routing protocols for the internet. The authors point out that it is well known that link dynamics and node removals, under the distributed Bellman-Ford algorithm (a distance-vector technique), can cause temporary or even long term looping behavior. Perkins and Bhagwat point out that almost all solutions proposed prior to DSDV involved the participation of inter nodal coordination to resolve these artifacts. These techniques, the authors argue, do not scale well to highly dynamic environments.
In order to address the looping behavior of the basic distributed Bellman-Ford algorithm, Perkins and Bhagwat propose a sequence number be associated with each destination in the distance vector. If the metric comes directly from contact with the destination, the sequence number is made even. If the metric is inferred due to link failure (perhaps caused by the node’s movement) then the sequence number is made odd. Nodes, when updating their distance-vectors as a response to this update, observe the sequence numbers and can therefore avoid the count-to-infinity problem (and subsequent packet looping behavior). The DSDV protocol is not without problems, however. In extreme dynamics the control overhead of broadcasting new metrics and sequence numbers can introduce a substantial control packet load. Minimizing control packet overhead was one of several primary goals of Gnawali et al. in the creation of CTP Noe.

To the best of our knowledge, the evaluation of the collection tree protocol variant (CTP Noe) by Gnawali et al. [38] has been the most rigorous testbed evaluation of any wireless sensor network collection protocol to date. CTP Noe emphasizes reliability, robustness, efficiency and hardware independence. We note that high throughput was not a primary goal of CTP, though it does appear to out-perform it’s peers (MintRoute [145], MultihopLQI) in this metric as well. In [38], the authors exhaustively evaluate CTP over an incredible 12 separate testbeds, seven hardware platforms, and six link layers, across both low and heavy external interference channels. Their work has made CTP with the 4 bit Link Estimator the premier routing protocol for most TinyOS applications. Extending concepts from Trickle [77], designed to push code updates into very large scale sensor networks, the authors implement exponential backoffs in beacon generation for
link estimation purposes. While links and routes remain stable, this exponential back-off tremendously reduces the overhead associated with broadcast-based link estimation. When transients occur (the authors cite the temporal fluctuations highlighted in [124]), the beacons reset to rapidly capture new link realities.

Citing looping behavior common of routing algorithms whose topology control mechanisms are orders of magnitude slower than data rates (MultihopLQI, DSDV), Gnawali et al. emphasize the dynamic parent selection of CPT Noe. Empirically, they find that CTP Noe nodes may select new parent nodes as frequently as every five packet transmissions for some networks. The authors work to avoid over-zealous re-routing, however. Being tree-based and single-path-route in it’s objectives, CTP Noe employs hysteresis in path selection. A node will only switch routes if routing metrics indicate the new parent node has an ETX at least 1.5 transmissions lower.

CTP Noe also includes a number of design decisions that are derived from lessons learned in prior literature. For example, the self-interference in a flow passing through a multi-hop network was cited in [79]. This phenomenon is avoided by CTP Noe by a transmission timer implementation, which reduces collisions with the subsequent incoming packet. The careful systems design of CTP Noe results in impressive performance across all twelve testbed environments, with delivery ratios in the 90% to 99.9% range in all tests. Additionally, the authors test CTP atop radio duty-cycling MAC, and successfully demonstrate asynchronous radio duty cycles as low as 3% with an aggregate network load of 25 packets / minute. This is an important achievement for long duration deployments.
2.4.3 Tiered Routing

Tiered network architectures, in networks composed of heterogeneous nodes, have been favored in a number of real-world wireless sensor network deployments (Great Duck Island [84], industrial monitoring [72], volcano monitoring [140], water pip leakage detection [125], and a redwood micro-climate survey [130], to name a few). The presence of a tiered architecture does not preclude any particular mote-level routing protocol, but simply indicates that the resources of devices in the network are not homogeneous. It is often the case that high resource nodes will use higher power network stacks (e.g. IP stack, socket programming) to ease programming.

Early theoretical works include a derived upper bound to deployment lifetime under optimal node role assignment [14], analysis of the tradeoff between execution time, accuracy, and energy consumption with heterogeneous node insertion and sub-task partitioning [73], and evaluating the sensing coverage area and sensing coverage degree under a cost-benefit analysis of heterogeneous node deployment [75]. In all works, strong performance of the heterogeneous networks sparked ongoing research into the challenges of heterogeneous node deployment and sub-task assignment.

Noting that a flat, large network has proven to not scale well in many real applications and that sensor fusion on the mote level requires highly custom code, Gnawali et al. create the Tenet Architecture [39]. By leveraging a master tier of 32-bit higher power processor nodes, Tenet supports easier code implementation of sensor fusion in-network,
while supporting greater scalability. This work shows that while application-specific sensor fusion code in the sensor mote tier reduces communications overhead, it does so at the cost of system complexity and reduced manageability.

The routing for Tenet is comprised of a number of existing and novel techniques within the tiered structure. Between master nodes, IP routing is leveraged. This allows master-level applications to use familiar socket programming techniques and IP mesh routing algorithms. Within the mote tier, routing toward the master tier is accomplished through metric beaconing and forest generation. Each node joins the parent mote having lowest metric, as this allows min cost access to the master tier. Once in the master tier, an overlay network assists in continued routing over the master network. The authors modify both MultiHopLQI and MintRoute to form the forest and support routing functions, MultiHopLQI is used in their experimentation. Finally, Tenet supports master-to-mote messaging. This is accomplished through a reverse-path adoption (akin to that employed by IP Multicast source-based routing) in the new forest-capable tree generation protocol. This data-driven point-to-point routing table requires one timer and one routing entry per child node.

Through a single pursuer, single evader pursuit-evasion game (PEG) implementation on Tenet, Gnawali et al. find that Tenet-PEG is more precise than Mote-PEG (a mote-only implementation) in localization of the evader thanks to it’s non-distributed approach. In their experimentation, the communications overhead of Tenet-PEG is lower than Mote-PEG, due to the ease with which they can implement an adaptive thresholding algorithm that reduced erroneous evader reporting.
2.4.4 Any-to-Any Routing with Low State Space

Early in the discussion of challenges facing very large scale ad hoc networks, researchers recognized that any-to-any routing would stress the resource constraints of these devices. As a result, a spectrum of solutions have been introduced, generally trading optimality (of the minimum cost path over the routing metric) for state space and computational load. We’ll now discuss some of the highlights in this field, as though we do not focus on Any-to-Any routing in the work presented here it is an important future effort. Further, understanding these alternative routing approaches is important when viewing the big picture of diverse routing alternatives.

Aiming for one extreme of the stretch / state space trade, geographic routing forwards packets hop-by-hop to the neighbor nearest the desired destination. In this manner, no state space whatsoever is required by the nodes in the routing framework, aside from per-neighbor geographic locations. A simple greedy protocol for geographic routing is proposed by Karp and Kung in [64], dubbed the Greedy Perimeter Stateless Routing algorithm (GPSR). One key challenge of geographic routing techniques is the handling of network holes, in which a node may find itself not the destination of a packet, but also discover that no neighbors are geographically closer to the destination. This traps the packet, and must be handled by the geographic routing protocol in some manner. The solution of Karp and Kung is to leverage local topology knowledge to route around the hole until once again a neighbor is discovered which is closer to the geographic destination. The very low control overhead and lack of need for coordination in computation of routing metrics allows GPSR to scale well in highly dense or mobile networks, but at a cost. Nodes
on the perimeter of network holes find themselves subject to higher traffic volumes, which can result in packet losses and node energy depletion.

Addressing this concern has been the focus of a number of virtual mapping techniques, an example of such is the Ricci flow application Sarkar et al. [116, 115]. By application of local gossip-style updates to virtual geo-addresses, the authors are able to achieve a virtual coordinate system in which no node finds itself without a neighbor closer to any destination (therefore no hole can result in the trapping of a packet). This simplifies the hole navigation heuristics, which could be foiled, used by [64]. Further, in order to restrict the loading difficulties on hole-perimeter nodes, the authors also leveraged conformal Möbius transforms to recursively virtual-map nodes to geo-locations inside the physically induced network holes. In this manner the holes in the network disappear within the virtual address space, obliterated by recursive virtual mapping. Clearly this comes at a routing stretch cost, which the authors explore.

Recognizing the inherent routing stretch tradeoff to zero-replication zero-state routing, Fonseca et al. introduce Beacon Vector Routing (BVR) [32] which aims to take the middle road of constant network-state. A subset of relatively few nodes are randomly designated as beacon nodes, to which the typical shortest-path routing trees are generated. Using the vector of distances to these beacon nodes, a distance metric is formed and then greedily routed over. While this method requires somewhat greater state information than geographic routing, it requires no node knowledge of location (which can be expensive in terms of node cost and energy consumption). Further, the authors of BVR demonstrate that geographic routing can be substantially sub-optimal in environments where physical distance is poorly correlated with link reliability. Differentiating BVR from a number of
the attempts to fix geographic routing is the emphasis placed on simplicity by Fonseca et al.

As there are similarities between pure geographic routing and the virtual coordinate system generated by BVR, they do share the weakness of greedy routing over a metric space containing holes. The novel solution used by BVR is two-fold. First, if pure greedy routing over the virtual metric fails (because no neighbors are deemed closer to the destination in the virtual space), the node forwards the packet to the beacon nearest the destination. Once arriving at this destination, the greedy forwarding resumes. Second, there still exists a possibility for greedy routing to fail even at the beacon nearest the destination. If this occurs, limited scope flooding is employed from the beacon. Thanks to distance metric knowledge, the node which is unable to make progress knows precisely how many hops away the destination is, and can therefore carry out a scoped flood. In simulations by Fonseca et al., the frequency of these events is found to be low in BVR, but the events substantially increase the transmission stretch when they do occur.

In a third technique, aimed at the middle-ground with respect to the routing space to routing stretch tradeoff, is compact routing described by Krioukov et al. [71] and subsequently demonstrated as a sensor network variant by Mao et al. [85]. In compact routing, a constant fraction of the nodes $O(\sqrt{N})$ are selected as beacons. Every node stores routing state for the beacons of the network, and each beacon node additionally stores state for the nearest $O(\sqrt{N})$ neighbors in its cluster. In this way, the routing state required consumes only $O(\sqrt{N})$ resources and the routing stretch is worst case bounded by a factor of 3.
The final approach to low state any-to-any algorithms is termed hierarchical routing, and is introduced by Kleinrock and Kamoun in [69], then extended to the concept of landmark hierarchical routing by Tsuchiya [131]. Though many variants of hierarchical routing have been proposed, they all share the foundation of these two works. Landmark routing can achieve any-to-any routing with as little as $O(\log N)$ state information per node. Nodes are organized into a multi-level hierarchy of clusters, each with its own cluster head (called a landmark). Nodes are labeled by describing the $O(\log N)$ clusters heads of which they are cluster members. One of the key challenges of landmark routing is the generation and maintenance of these multi-level clusters, which prompted concern that the complexity would be too great for low resource Wireless Sensor Networks. Further, landmark routing on some graph topologies can result in arbitrarily bad routing stretch. Performance evaluation in both TOSSIM and on a real 60 node deployment by Iwanicki and Steen [55] found that though there exist graph-theoretic topologies for which the routing stretch can be quite bad in hierarchical routing, the realities of physical node placement and short radio range generates graphs that are geometric. That is, the average internode distance grows as $N^v$ for some $v > 1$ and node count $N$. In these geometric graphs, it is shown that the stretch of hierarchical routing can be close to 1 [33, 69].

2.4.5 Routing Under Mobility

The body of related literature discussing topics associated with routing under node mobility, relay mobility, or sink mobility is extremely vast. The topic has held quite a lot of interest in the broader wireless community, and a thorough discussion of all relevant
publications would be burdensome here. We therefore will focus on the motivation, initial collection-centric theoretical results, and system protocols designed with mobility in mind.

In looking for alternative solutions to the capacity scaling challenge posed by the work of Gupta and Kumar [41], researchers also began to investigate the potential benefits of node mobility. In the landmark study by Grossglauser and Tse [40], using a SINR model and under uniform node mobility it is shown that the capacity of mobile networks can remain constant. While not as positive as the cooperative results, this technique requires no modifications to hardware. In order to achieve constant throughput scaling, Grossglauser and Tse leverage exclusively two-hop routes between source-sink pairs. In this manner they leverage route (and spacial) diversity. Unfortunately, their uniform movement about a unit disc graph is somewhat limiting and unrealistic.

Bringing the discussion to a wireless sensor network context, Shah et al. proposed data mule [118]. Their three-tier network design consists of sparsely deployed resource poor sensor motes which relay data to intermediate nodes having moderate resources, called mules, which in turn are subject to uncontrolled mobility that allows offloading of data to base stations. In this architecture, by accepting potentially high delays, all data is collected in at most three hops. The energy consumed to collect the data is therefore very low, but the buffering required can be quite large, potentially resulting in data losses.

Returning with a new outlook on the Data Mule work of Shah et al., Wang et al. consider mobile nodes with rich resources that are able to relay data in order to assist the surrounding low power network [137]. This concept is known as a mobile data relay, and though Wang et al. demonstrate that a mobile sink will always outperform a collection of
mobile relays (assuming a trivial number of relay nodes), in the absence of a mobile sink this new role for heterogeneous nodes is worthy of consideration. The authors demonstrate that adding only a few resource rich mobile nodes can provide the same performance gains, with respect to network lifetime, as increasing the network density several times (and then sleep duty cycling nodes).

The Data Mule concept and extensions by Wang et al. can be seen as precursors of a new data collection paradigm, now called participatory sensing. Introduced by Burke et al. in [20], participatory sensing emphasizes the value of user collaboration in city-scale or larger sensor networks. The authors note the positive scaling properties of user volunteered sensing and collection, specifically in a new world where smart phones are owned by millions of residents. These smart devices can accurately timestamp and geo-code sensed measurements or collected data, and often have multi-band radio technologies, capable of supporting data collection over bluetooth while simultaneously streaming it to a database over the cellular backhaul.

An important differentiating property of mobile routing research is the periodicity of movement for the sink or relay. In slow movement regimes, it is often assumed that the mobile device will pause for long periods, while not taking long to transit between stationary points. This might be thought of as sink re-location. In this setting, the routing protocol needs to build new routing paths with overhead that is justified by the frequency of sink re-location. In the following discussion, we note that both Hyper [117] and MobiRoute [83] are relatively slow movement targeted system protocols. In fast movement regimes, the mobility is constant and relatively rapid. In this regime, re-building routing structures every time a node moves is of prohibitive system cost.
Solutions that target fast movement include Data Mule [118], the directed-diffusion work by Intanagonwiwat et al. [54] and Kansal et al. [63], and generally applications in the participatory sensing space [20].

Having motivated mobility in wireless sensor networks, we would now like to highlight some of the prior work for WSN having mobile sinks. Hyper, of Schoellhammer et al. [117], is a tree-centric protocol with the goal of supporting mobile sinks within the wireless sensor network. An enhancement termed fast beaconing allows for fast sink discovery and link estimation when the sink moves into a new region of the network. In the author’s stated objectives, their goal is to support sinks that move at most every thirty seconds. Experimental data in [117] suggests that a five-hop topology requires approximately one second to converge on a new tree after sink relocation.

Another sink mobility routing protocol for WSN, MobiRoute, has been proposed by Luo et al. [83]. A key differentiator from Hyper is the assumption that the sink node has control over its mobility. MobiRoute is written by extending MintRoute [145] to support slow sink mobility. As MintRoute uses a distance-vector approach, control overhead and packet delivery reliability become problematic in extremely mobile networks. To minimize control overhead and increase delivery ratios during slow mobility, MobiRoute leverages explicit sink handoffs.

2.4.6 Cooperative Routing

In communications, the term “cooperative” encompasses results at multiple layers of the networking stack. A number of these techniques take place at the link layer, generally requiring customized hardware and higher cost radios than typically found on inexpensive
wireless sensor network nodes. We’ll omit these concepts as they were not applied in our work and their implementation is generally orthogonal to the routing we discuss here. In this section we will discuss receiver diversity and network coding results, both of which require no radio-level modifications.

Leveraging receiver diversity for routing improvements was first proposed by Larsson in [74]. He terms it *Selection Diversity Forwarding (SDF)*, and through a few simple traffic models and Monte Carlo simulations demonstrate strong motivation for a new view of wireless channel broadcast capability and reception variability. Larsson suggests for the first time that the naturally unpredictable broadcast nature of wireless is potentially a benefit that can be leveraged opportunistically. Specifically, a node using SDF determines a set of candidate next-hop nodes whose routing metric is of lower cost than its own. The number of candidates might be in the range of 2-4, depending upon link quality and data payload size. The node then broadcasts the packet, and waits to hear acknowledgements from nodes within the candidate list. Finally, the send node determines the optimal receiver and sends a grant message, all non-granted nodes drop the received packet.

A key open question of Larsson’s work was the overhead and complexity challenges of this four-way handshake. These concerns were addressed by the design and evaluation of the ExOR protocol by Biswas and Morris [16]. In the design of ExOR, the authors decided to focus on bulk packet delivery. This decision was made for several reasons, including a reduction in the overhead associated with the receiver diversity mechanism, to support gossip-based propagation of per-packet transmission progress within the batch, and to allow for a hybrid approach which reverts to classic single-path routing techniques once 90% of the bulk transfer has successfully been collected through cooperative means.
ExOR has been implemented in Linux routers of Roofnet, the 38-node 802.11b mesh network that covers approximately six square kilometers of Cambridge, Massachusetts. In experimental evaluation between 65 node pairs, Larsson demonstrates a 3-fold increase in maximum throughput for the median pair (11KB/sec for traditional routing, 33KB/sec for ExOR). In looking at the throughput distribution, he generally found that nodes that were further apart (multi-hop) yielded greatest speedup by ExOR. This makes intuitive sense, as ExOR leverages fortuitous long-range receptions to accelerate the batch delivery.

There are a few important considerations to briefly discuss with respect to the ExOR results and implementation. First, in order to coordinate batch forwarding and minimize un-necessary re-transmissions of packets by lower priority nodes, Biswas and Morris tie the routing and MAC scheduling by imposing a strict access schedule on the routers. Second, the authors note that the throughput of successive packet batches sometimes varies substantially. This is attributed to the opportunistic leveraging of intermediate quality links by ExOR, which may at times perform poorly over the time scale of a packet batch, and has implications for delay-sensitive applications. Third, it is important to note that because ExOR pre-simulates the network to prioritize nodes for reception, the source must have complete knowledge of the ETX metric for all links in the network. This weakens the ability of ExOR to respond to rapid topology dynamics.

Before discussing the follow-on work to ExOR that address some of these challenges, we need to discuss the concept of network coding. Network coding is first described in the landmark paper by Ahlswede et al. in [3]. In essence, the goal of network coding techniques is to combine packets as they arrive at intermediate nodes within the network, then forward the output of these packet combinations. Once arriving at a destination,
records in packet headers allow the node to re-construct the original data once sufficiently diverse combinations have arrived. The question then, in part, is how complex this packet combining and decoding would become. A flurry of follow-up work [80, 70, 56] demonstrates that for multicast applications, linear network coding can achieve the maximum network capacity while consuming only polynomial time resources for the purpose of encoding at the intermediate nodes. Further easing the complexity, Ho et al. extended these works to demonstrate that the properties held even under randomized coefficients [46]. In aggregate, these are impressive results. Random linear network coding is actually quite simple for intermediate nodes to accomplish (a potentially important criteria for these networks, as intermediate nodes dominate in number and may be resource limited).

Returning to the drawbacks of ExOR discussed above, and noting that the cooperative technique of random network coding could augment the weaknesses of ExOR, Chachulski et al. propose MAC-independent Opportunistic Routing and Encoding (MORE) [21]. By randomly mixing packets before transmission, routers no longer need to impose a strict radio scheduling prioritization for routers in the network. Specifically, MORE implements and demonstrates on a testbed a low-complexity algorithm for intra-flow wireless network coding. In another advantage over ExOR, no global network knowledge was required by the source to prioritize forwarding nodes. The randomized mixing makes this completely un-necessary. By omitting this pessimistic scheduling and allowing more opportunistic spatial diversity, Chachulski et al. find experimentally that MORE out-performs ExOR by 22% for the median unicast flow, and as much as 45% for four-hop flows. Further, MORE excels at multicast support, scaling per number of destinations to as much as 300% the throughput of ExOR.
While addressing some of the weaknesses of ExOR for realistic usage, the MORE protocol still requires bulk data delivery for efficient operation and suffers from relatively dynamic packet throughput. The variability may be traded for packet block size, but this comes at a latency cost in smoothing the throughput. Effectively moving the delivery delay out of the time scale of link variability. This is a property of all network coding approaches. Second, though MORE uses algorithms that are relatively low complexity, they are still un-demonstrated on light-weight sensor network nodes. In the experimental evaluation of MORE [21], the authors use software defined radios and 800 Mhz CPUs. These resources are several orders of magnitude more substantial than today’s low power wireless sensor nodes. Third, like ExOR, due to overhead and latency concerns MORE targets medium to large transfers (i.e. 8 or more IP packets, presumably 1,500 Bytes each). The authors argue that smaller packets are better delivered by standard best path routing, with which MORE happily co-exists (in fact the packet batch ACKS are sent by shortest path to the source using this traditional routing mesh). Finally, we note that MORE is designed for stationary wireless mesh networks.

The exceptional multicast performance of MORE can be attributed to the intra-flow network coding techniques it applies. Inter-flow network coding is a much more difficult problem, we refer the reader to COPE by Katti et al. [66] for the latest proposals in inter-flow network coding. We feel discussion of inter-flow network coding lies outside the scope of this thesis, provided it’s current lack of maturity and dis-similarity to our own work. Even without inter-flow mixing, the performance of MORE can be improved upon. This was made clear by the MIXIT protocol and experimental evaluation by Katti et al. [65, 66]. The core feature recognized and leveraged by MIXIT is that there is PHY
information being discarded by the broad packet rejection mechanisms of typical reliable MAC protocols. If some level of confidence for sub-portions of packets could be distilled from the radio, it would become possible to apply random linear network coding on a sub-packet scale, therefore maximally putting to use the successfully received data. This is precisely the goal of MIXIT.

In describing MIXIT, the authors term the technique symbol-level network coding (where here a symbol is not to be confused with its PHY layer definition, a symbol may be multiple bytes in length within MIXIT). Intuitively, the greater the protocol’s ability to tease out symbols that are correctly received, the greater the expected progress in relaying data to the destination. There were substantial barriers to implementing such a minute-scale network coding, particularly the overhead and necessity for optimization associated with describing which pieces of packets were being forwarded and subjected to random linear network coding. Specifically, MIXIT uses run-length encoding to describe the network coded linear multipliers and their associated batch positions. This run-length encoding is further reduced in overhead through a novel dynamic programming approach which promotes innovative linear combinations while reducing per-mixing run length count and therefore MIXIT packet overhead.

Another key difficulty in a real system implementation lies in correcting for erroneous pieces of packets that are incorrectly assigned reliable belief status by the PHY layer. MIXIT employs a symbol-level network code that is also an end-to-end rateless error correcting code. This error correction scheme supports error detection, correction, and proper packet re-constitution at the sink. Somewhat incredibly, the error-correcting code is not affected even if all received symbols are corrupted. In fact, if \( m \) erroneous symbols
are incorrectly classified as clean, the destination needs only $B + 2 \cdot m$ symbols to recover
the original $B$ symbols. The authors point out that this guarantee is theoretically optimal.

In the queue awareness of their credit-based probabilistic forwarding mechanisms, MIXIT
applies heuristic techniques similar to backpressure routing. A broadcasting node
assigns forwarding responsibility to receiving nodes by allocating credits. The authors
define a C-ETS routing metric, which is a weighted combination of symbol-transmissions
to the destination and the current queue occupancy of the node in question. The neighbor
with lowest C-ETS is assigned a credit of 1.0, representing a 100% chance of incorporating
the broadcast packet if received. The second highest C-ETS neighbor is assigned credit
$(1 - p_1)$, where $p_1$ is the symbol reception probability of the highest ETS neighbor.

Intuitively, the authors are trying to ensure that the proper number of symbol replicas are
forwarded in expectation, based roughly upon symbol loss rates of neighbor nodes. Using
probe mechanisms, MIXIT periodically measures these symbol reception probabilities.
The authors note that 30% of the empirically determined capacity enhancement can be
attributed to the reduction in queue drops due to queue-aware forwarding. Finally, it is
important to note that the stated goal of MIXIT (per Katti et al.) is to support bulk
transfer over static mesh networks. Their periodic symbol reception estimation, batch
processing, and network coding algorithms become inefficient in non-bulk settings, and
in settings with dynamic network topology.

2.4.7 Potential Routing

Potential routing solutions assign costs to nodes in the network, using graph-theoretic
principles to iteratively update the values of these costs. Packets are then routed to the
neighbor having least potential. One of the earliest such algorithms to be decentralized in computation was pioneered by Gallager [34] and subsequently improved through step size adaptation by Bertsekas et al. [13]. Their algorithms use adaptations of Newton’s Method to minimize system average traffic delay while solving a multi-commodity flow in which delay is a convex function of traffic rate. These algorithms and subsequent variants require a-priori knowledge of the long term average traffic demands on the network, and operate on an invariant network of constant capacity links. They are therefore more restricted. Broadly, these first and second derivative Newton-Method based algorithms are called steepest gradient search algorithms.

More recent work by Basu et al. [10] uses these steepest gradient search methods to assign potentials to nodes that are a function of both cost to the destination and queue congestion. Their PBTA routing protocol iteratively converges on these potentials, which are then routed on. The authors prove that the queue sizes will remain bounded, and that looping cannot occur under PBTA. The underlying traffic assumptions on which the proofs rely are rather strict, and require rapid node potential updates with respect to queue updates. Additionally, the queue sizes required for stability are still quite large, as the notion of congestion is path based, not next hop.

2.4.8 Backpressure Routing

Recent developments in stochastic network optimization theory have yielded a very general framework that solves a large class of networking problems, ranging from flow utility maximization [91, 29], energy minimization [91, 99], network pricing [51] to cognitive radio applications [132]. Among the approaches that have been adopted, the family of
Backpressure algorithms [37] are recently receiving much attention due to their provable performance guarantees, robustness to stochastic network conditions and, most importantly, their ability to achieve the desired performance without requiring any statistical knowledge of the underlying randomness in the network. To date, there has been no systems implementation of the dynamic backpressure routing component of these algorithms. In the following section we will give special focus to the work leading up to and extending the backpressure algorithm described by [37], as it forms the foundation for our work.

### 2.4.8.1 Theoretical Work in Backpressure Stacks

The intellectual roots of dynamic backpressure routing for multi-hop wireless networks lie in the seminal work by Tassiulas and Ephremides [128]. They considered a multi-channel downlink with ON/OFF channels, and proved that using the product of queue differentials and link rates as weights for a centralized maximum-weight-matching algorithm allows any traffic within the network’s capacity region to be scheduled stably. The result was subsequently generalized to multi-rate transmission models and systems with power allocation [7],[98],[37].

In [98], Neely et al. build upon the max-weight work of Tassiulas and Ephremides to support a general power control problem for multi-beam one-hop satellite links. This work is extended substantially in [99], where Neely et al. make several novel contributions that lay the foundation for many future publications by providing joint power allocation and throughput optimality in multi-hop networks while supporting links having generalized inter-link interference and time varying channel capacity. This generalizes the results of
Tassiulas and Ephremides. Neely et al. define a concept of network capacity, different from the information-theoretic concept of capacity. They then bridge the existing gap between network capacity, throughput optimality, and network optimization. Their work applies to multi-hop wireless networks with general ergodic arrival and channel state processes, and need not know the parameters of these processes. The authors assume a rate-power curve is known for each link, possibly influenced by other transmission power decisions (e.g. an SINR model). They describe the Dynamic Routing and Power Control (DRPC) algorithm and its power allocation and routing/scheduling control decisions, which they prove are throughput optimal while obeying per-node power budgets. Finally, Neely et al. provide analytic bounds on the asymptotic time average delay experienced by packets traversing a network under the DRPC algorithm.

The power control work is subsequently extended by Neely in [92]. Here, through the introduction of a tuning parameter $V$, Neely is able to maintain throughput optimality while coming arbitrarily close to the optimal (minimal) time average power consumption per node ($\bar{p}_i^*$ for node $i$). Increasing $V$ results in the time average node $i$ power consumption approaching $\bar{p}_i^*$ like $O(\frac{1}{V})$, while the queue size bound grows like $O(V)$ and therefore the delay bound grows like $O(V)$. This and another work by Neely et al. [97] are the first applications of Lyapunov drift for the joint purpose of utility or penalty optimization and throughput optimality. The authors term this energy efficient, throughput optimal algorithm the Energy-Efficient Control Algorithm (EECA).

Also in [92], Neely introduces the concept of virtual queues within the Lyapunov drift minimization framework. Leveraging this novel concept, he is able to support time average penalty or utility constraints. Specifically, in [92] Neely notes that one might
relax the power minimization objective and instead specify per-node time average power consumption constraints, then maximize network capacity subject to these time average constraints. These additional virtual queues are serviced at the constrained energy rate, while arrivals are equal to the per-timeslot power expenditures of the node. In order to maintain stability in these virtual queues, which are introduced into the Lyapunov network in [92], the virtual queues must also be strongly stable.

The algorithms so far discussed have all made control decisions at the Networking layer and below. The external arrival processes (exogenous data) were assumed to be outside the control of the algorithm. The arrival rate was either within the network capacity region, resulting in stable queues but possibly substantial delay, or it was outside the network capacity region and queues grew without bound. Noting this and seeking to introduce source rate control into the previously laid foundation, Neely et al. extend the capabilities of the analytical framework into the transport layer through the introduction of per node packet reservoirs from which admission control takes place [97, 91]. The resulting family of Cross Layer Control (CLC) algorithms perform an additional function of flow control, in addition to scheduling, routing and resource allocation. Again leveraging a tradeoff parameter $V$, the CLC1 algorithm maximizes (minimizes) a non-decreasing and concave utility (penalty) function subject to throughput optimality. Similar to the EECA algorithm, the CLC algorithm approaches the per-node minimal optimal utility (penalty) $\bar{p}^*_{au}$ like $O(\frac{1}{V})$, while the queue size bound grows like $O(V)$ and therefore the delay bound grows like $O(V)$.

The union of enhancements including multi-rate time variable channels [98], the concept of network capacity [99], utility or penalty minimization [92], source rate control
and utility or penalty time average bounds have now yielded a rigorous framework for a wide variety of stochastic network optimization problems. We note that the foundational work established in are collectively available in a book by Georgiadis, Neely and Tassiulas.

The receiver diversity concepts previously discussed in Section 2.4.6 are applied to stochastic network optimization by Neely and Urgaonkar. Their DIVBAR algorithm, which jointly maximizes network capacity while minimizing asymptotic time average power, leverages receiver diversity. The control decisions of their DIVBAR algorithm carefully avoid packet replication by explicit packet responsibility handoff. Packet replication is a feature of most if not all of the algorithms discussed in Section 2.4.6 (e.g. ExOR, MORE, MIXIT). The approach of packet replication by these protocols, implemented in real systems, has been motivated by concerns of overhead and errors in the three-way handshake required for explicit packet responsibility handoff. By avoiding packet replication, Neely and Urgaonkar are able to give analytical guarantees of throughput optimality under the Lyapunov drift analytical framework, but systems implementation remains an open question. In order to simplify and distribute DIVBAR control decisions, in the authors first assume orthogonal channels without inter-node interference. They then extend the DIVBAR algorithm to channels with inter-node interference, note the existence of complicated commodity selection requirements, and propose a distributed and constant-factor approximation algorithm.

While we have focused on literature pertaining to backpressure stacks approached from a Lyapunov drift minimization framework, we note that a parallel track approach has been developed by Stolyar and subsequently extended to the wireless domain.
by Akyol et al. [4]. Their work has developed a related backpressure-based stochastic optimization framework using duality and gradient descent techniques.

Though backpressure stacks have a number of promising characteristics, there are barriers to adoption that exist. We will next discuss these challenges, which include delay performance, queue growth (complexity) in any-to-any usage, network scalability when faced with finite queues, and complex link scheduling coordination.

### 2.4.8.2 Addressing Delay in Backpressure Systems

Though known to be throughput optimal, the delay characteristics of the max-weight algorithm has been explored more thoroughly only recently. Neely provides a survey of delay results in [95], relating these results to a single-hop N-multi-channel downlink / uplink. In [98] and [37], it is shown that the asymptotic average delay experienced across an N channel downlink is bounded by a function $O(N)$. Neely points out that this bound is tight when arrival and channel processes are correlated, but in [95] proves that for ON/OFF channels the max-weight policy yields asymptotic delay that grows $O(1)$ with N under independence assumptions. That is, max-weight is order-optimal in this setting. Further, Neely provides an example multi-rate channel system for which asymptotic average delay must grow linearly in N. Therefore demonstrating the fundamental delay disparity between ON/OFF and multi-rate channel scheduling. Addressing the delay challenge has therefore been of substantial importance.

We note that in recent systems-centric delay reduction literature, there is emphasis on two causes of delay in these systems. First, there is research focused on the delay incurred at low data rates by looping or meandering packets. When the system is not
loaded near the edge of the capacity region, there is no goal embedded within traditional backpressure routing to reduce this indirect routing. Second, there is delay incurred by long-standing backlogs, particularly when using utility optimization frameworks. These standing queues result in long per-packet delivery delays even when shortest paths are taken. We will now describe these separate contributions.

Reducing Path Discovery Impacts: The delay bound derived in [99] scales inversely with network load. Counter to most routing techniques, this implies that the per-packet average delay will actually increase as load decreases, for sufficiently low load. This fact is noted by Neely et al., who propose that a per-node bias be introduced into the link weight calculations. In this manner the authors preserve throughput optimality at high load, while enhancing delay performance for low loads. The Enhanced Dynamic Routing and Power Control (EDRPC) protocol [99, 91] is therefore proposed. Per node biases are computed offline, through an external routing metric mechanism, or by time-average backlog values at moderate to high load, then applied per-node to bias the backpressure toward the destination. We note that these weights must be determined either externally or through a learning process and may form an Achilles heel to what is otherwise a very robust network protocol.

In recent work by Ying et al. [149], a specialty queue expansion is proposed which supports strict per flow packet hop count constraints. Importantly, the algorithm does not only guarantee a time average (as is the case with virtual queues used in [91]) but guarantees per-packet non-violation. This comes at a cost of queue count expansion, which the authors argue can be mitigated using the queue clustering techniques of [150]. The authors prove that the algorithm is stable for any stabilizable traffic rates subject
to the per-flow constraints. Simulations of the algorithm demonstrate a reduction of per-packet delay of 333x on a sample network topology. The algorithm requires shortest-path computation external to the backpressure algorithm, which has potential to weaken the responsiveness to network dynamics. Additionally, the algorithm does nothing to reduce the delays caused by standing backlogs. The theoretical work of Ying et al. [149] can support the substitution of ETX minimization for hop count optimization, though this was not tested in their simulations.

**Addressing Queue Backlogs:** Naghshtvar and Javidi demonstrate throughput optimality of a wider class of Lyapunov functions within the Lyapunov Drift framework [90], and propose an algorithm that computes next-hop backlog as a product of min-congestion-path to the sink and link cost. The authors construct a pathological worst-case topology for quadratic Lyapunov networking. In this configuration, a large pool of highly interconnected nodes, having only a single link exiting the cluster, frequently traps packets sourced outside the cluster. Their simulations show drastic delay improvement for worst-case topologies under the traditional quadratic Lyapunov function. To date, these modified queue backpressure techniques have not been demonstrated in systems.

In novel work leading to an alternative delay reduction technique, researchers have formulated backpressure solutions to the single and multi-rate wireless multicast problems [19]. In their solution, shadow data is sourced by multicast destinations and scheduled / routed by a backpressure framework of shadow queues toward the multicast sources. Real data packets sourced by the multicast source then follow the reverse path of the shadow data, consuming link allocations proportional to the shadow data scheduling on the reverse link. This concept of shadow data and shadow queues providing transfer
permissions was subsequently extended by Bui et al. to support reduced delay in unicast backpressure implementations [18]. The key to delay reduction here is that forwarding queues need not build up in order to promote forwarding, as the shadow queues serve this role (and are fed by shadow sources which upper bound the real data rates). The challenge to real system implementation is that backpressure stacks operating on shadow data must know link rates, but calculating link metrics in real systems requires realistic inter-node interference at the rate of interest.

In [18], the authors analytically prove that for a line topology with $N$ nodes optimizing for proportional-fair utility or max-min fair variants, the aggregate queue backlog ($O(N^2)$ in traditional backpressure) is reduced to $O(N)$ under the shadow data routing technique. The authors then simulate the shadow algorithm for line and grid topologies under various shadow-data rates, again under fixed routes per flow. The authors then migrate to a discussion of dynamic routing and propose the min-resource routing modification, in which their backpressure framework jointly minimizes inter-queue transfers and maintains throughput optimality, while running the shadow algorithm.

**Addressing Service Latency:** Aside from the system-viewpoint sources for delay, there is also work to tighten the theoretical bounds on average packet delay. The crux of the bound looseness is the following: If queues are not sufficiently backlogged to leverage the stochastic channel capacity during times of plenty, the system performance suffers. Edge events are defined as forwarding queue underflows, in which channel capacity exceeds current forwarding queue storage volume. We’ll now discuss the related work on service latency minimum bounds.
Berry and Gallager prove in [12] that the best possible energy-delay tradeoff is \( O\left(\frac{1}{V}\right) / O(\sqrt{V}) \) over a fading channel, if no packet drops are allowed. They describe a transmission power and scheduling algorithm which achieves this bound, and uses a counter-intuitive new drift mechanism. While most bounds had made it their goal to have negative drift regardless of queue occupancy, Berry and Gallager implement a positive drift for queues with less than \( O\left(\frac{1}{V}\right) \) occupancy, and a negative drift for queues greater than \( O\left(\frac{1}{V}\right) \) occupancy. This has been referred to this as buffer partitioning. The queue therefore settles around a backlog of \( O\left(\frac{1}{V}\right) \). This tightens their bound in that they are able to better leverage good channel characteristics when they arise by sending larger bursts of data, tightening their energy-delay bound.

Using buffer partitioning modifications to performance optimal Lyapunov networking, Neely proves in [93] that for overloaded systems, requiring source rate control for stability, a utility-delay tradeoff of \( O\left(\frac{1}{V}\right) / O(\log(V)) \) is achievable. Neely replaces the quadratic Lyapunov function of [97, 91, 37] with a mixed Lyapunov function that promotes buffer filling to an equilibrium of \( O\log(V) \), and necessarily includes virtual queues introduced in [91] [78] to maintain throughput optimality. The mixed Lyapunov function is exponential in data queue size and quadratic in virtual queue size.

Revisiting the energy-delay tradeoff, in [96] Neely proved new lower bounds for the Berry and Gallager paper [12] provided some packet drops are allowed. Using the mechanisms of [93], he shows that the optimal energy-delay tradeoff can be reduced to \( O\left(\frac{1}{V}\right) / O(\log(V)) \). Neely provides a joint intelligent packet discard policy and power allocation scheme that achieves this bound for any arbitrarily small dropping ratio. Similar to the work of Berry and Gallager in [12], Neely constructs a buffer partitioning. However,
Neely's buffer has fixed size, and discards arrivals that burst above this threshold. In doing so, he proves that the delay bound can be reduced while dropping only an arbitrarily small fraction of packets. Again, Neely employs mixed Lyapunov function that is exponential in data queue backlog and quadratic in virtual queues. Finally, in [96] a novel online algorithm to adapt the data queue drop threshold $Q$ is discussed, so as to reduce the frequency with which the channel capacity is unsatisfied by the queue backlog.

2.4.8.3 Network Scalability

Supporting finite data queues: On the topic of scalability, most foundational work in backpressure stacks assumed infinite node storage resources. There have been a number of efforts to address this deficiency in recent years. In [93], Neely demonstrated that in a one-hop setting his super-fast dynamic control algorithm could in fact operate with a finite data queue. Similarly, through application of an online algorithm in [96], Neely demonstrates that by allowing some packet drops to shape the packet delay, a finite data queue is supportable. Similar to [93], these results apply specifically to multi-channel, multi-user single hop topologies.

Additionally, we note that the queue gravity results and associated Fast Quadratic Lyapunov Algorithm (FQLA) proposed by Huang and Neely in [50] can easily be modified to support finite data queue and bounded packet losses for multi-hop networks. The strength of all three results [93, 96, 50] is in their ability to bound the drop rate incurred by the finite data queue insertion. Limiting their realistic usage is the fact that they cannot yet be applied to multi-hop networks [93, 96] or that they require statistical knowledge or learning of network parameters [50].

51
In a work discussed above for the delay reduction contributions, we note that the shadow algorithm proposed by Bui et al. [19, 18] can support finite data queues to a degree. Because the backpressure algorithm operates on shadow data, which will not be dropped, data packet losses (if they occur) will not impact the routing decisions of the shadow algorithm. The results of [19, 18] do not provide bounds on loss rates introduced by an introduction of finite data queues, as finite data queue support is not discussed in their work.

**Per source-sink queue complexity:** The backpressure routing stacks proposed in [92, 91] require maintenance of per-destination queues. This results in storage requirements that grow $O(N)$ with $N$ the network node count.

Returning again to the work of Bui et al. [18], the authors discuss the queue complexity gains possible through the shadow algorithm. They note that the data queues may be maintained per-neighbor (a tremendous complexity reduction versus per-destination) *if routes are static*. The authors conjecture that it may be possible to extend their techniques to support such queue complexity reduction even in dynamic routing.

The conjecture was proved correct by Athanasopoulou et al. [9] in recent efforts that have brought together the multiple techniques pioneered by Bui et al. in [19, 18]. Athanasopoulou et al. [9] describe a novel backpressure-based per-packet randomized routing framework that runs atop the shadow queue structure of [19] while minimizing hop count as explored in [18]. Their techniques reduce delay drastically and eliminates the per-destination queue complexity *even while supporting dynamic routing*, but does not provide tighter analytic bounds on average packet delay.
Throughput Optimal Scheduling

In our work, we have elected to explore the advantages of backpressure routing. As such, we have not employed any of the recent MAC layer optimizations. This decision was in part motivated by our own prior research in the Backpressure Rate Control Protocol (BRCP), in which we employed a number of neighborhood max-match mechanisms and found no improvement in the network capacity [121]. Despite our design choice, it is important to highlight that advancements in MAC protocol design have occurred. MAC Layer improvements supporting backpressure routing are an important (and large) future extension for our work.

The core challenge to direct implementation of backpressure routing lies in computing a max weight independent set every timeslot, a task that is NP-Hard. A number of approximation techniques have been explored. Recently, the Greedy Maximal Scheduling (GMS) algorithm has been of focus for first switch arbitration [27] and then link scheduling in multi-hop backpressure approximation algorithms [62]. The local pooling condition, defined by Dimakis and Walrand [27] provides insight into the performance of Greedy Maximal Scheduling (Longest Queue First). This is subsequently extended by Joo et al. [62] who defined the efficiency ratio of GMS through a graph property they termed the local-pooling factor. Joo et al. provide an iterative algorithm to compute the efficiency ratio of GMS, and prove that it achieves the full capacity region for tree topologies under the K-hop interference model.

Though computationally simpler, GMS is still either centralized, or distributed and iterative [47]. Neither is ideal for rapid scheduling of wireless links. More attractive
solutions involve optimizing CSMA or CSMA/CA algorithms to approximate maximum weight match objectives. In [60], Jiang and Walrand provide a distributed algorithm to adaptively modify the CSMA backoff parameters. The authors prove that this CSMA MAC is throughput optimal provided certain assumptions, such as perfect channel sensing, and no collisions. Jaing and Walrand later removed this collision-free assumption in [61].

More recently, Ni and Srikant note that in [102] that prior work in CSMA throughput-optimal parameter adaptation results in excessive delays when compared with traditional CSMA heuristics. They propose distributed discrete time randomized algorithms that support throughput-optimality while include heuristics to improve delay performance.

Finally, Akyol et al. [4] derive algorithms and carry out simulations that demonstrate the potential for TCP traffic performance gains through modifications that build upon the framework of 802.11. The authors assume a single communications channel, with fixed flow count and fixed routing (OLSR or AODV). Though their results are encouraging for real systems deployment of similar backpressure supporting MAC modifications, their fixed routing assumption makes the work inapplicable to our goals here. Their resulting MAC prioritization algorithm is similar in nature to the DiffQ system implementation, which we will discuss in section 2.6.2.

2.5 Source Rate Control

We will now discuss the objectives of source rate control, select existing literature for rate control on quasi-static routing trees, and preliminary implementations of source
rate control using backpressure techniques over static routing topologies. Source rate control has been a more recent emphasis, largely because these activities typically occur at the transport layer. The WSN research community first had to develop reasonable performance data link and network layer protocols. Recent experimental researchers have discovered that the interactions between network layers is very important, as unintended consequences of interface and protocol design can be catastrophic. Choi et al. [23] cite one recorded example in which researchers at Delft University observed negative interactions between Deluge [52] and MintRoute that caused network collapse and a 2% packet delivery ratio. Further, the volcano monitoring deployment By Werner-Allen et al. [141] required substantial MultiHopLQI modifications in order to support the multiple services running simultaneously in their network. Results such as these motivate Choi et al. to suggest the necessity of network protocol isolation in [23].

2.5.1 Source Rate Fairness Objectives

In most settings, a throughput maximizing utility function ($\sum r_i$) is grossly unfair within wireless sensor networks. Due to the shared wireless medium, this utility function will always be maximized by emphasizing exclusively single-hop data collection to the greatest extent possible, only pulling data from multi-hop nodes if the single hop source rates are insufficient to fully consume the wireless channel capacity.

Looking to enable a concept of fairness, the max-min fair objective was defined. Here, given a vector of source admission rates ($\vec{R}$), the least index in the vector has been maximized. In settings with inelastic demand, this utility function aims to give every
user an opportunity for non-zero utility from their network allocation. Variants of max-min fair include weighted max-min fair (in which the weighted source rates are optimized so as to maximize the least element) and lexicographical max-min fair (in which the sorted rate vector $\vec{R}^*$ is index-wise strictly greater than all alternative rate vectors). The work of Radunović and Boudec [111] gives a generalized max-min framework and surveys the micro-economic origins of the max-min variants.

We will also discuss a flavor of source utility function termed proportional fair. Researchers at Qualcomm wished to leverage multi-user spacial diversity in cell tower downlinks, but found the previously defined user utility functions to be unhelpful analytically. As a solution, Jalali et al. [58] defined a proportional fair source rate controller that maximizes $\sum_i \log(r_i)$. Following the brief exposition in [58], further evaluation was accomplished by Viswanath et al. in [134]. The proportional fair utility function has since migrated into multi-hop network optimization [37], and has been extended to support join elastic and inelastic demand [8].

The max-min fair objectives can be difficult to work with in some theoretical frameworks. Contributions by Mo and Walrand propose a solution leveraging a tunable source rate utility function, now known as alpha-fair [87, 86, 110, 127]. Through selection of an alpha parameter, the utility function can represent proportional fair, or asymptotically approach max-min fair.

It is important to consider the trade-offs between these alternative utility functions. As was mentioned above, the throughput maximization objective is grossly unfair in wireless networks. At the opposite end of the fairness / throughput spectrum, max-min is widely considered to be the fairest of utility functions [110]. In fact, Radunović and Le...
Boudec describe in [110] a fairness index which measures source rate deviation from the max-min fair allocation (their fairness index is a mild variation of Jain’s metric in [57]). The authors investigate the performance of max-min in wireless ad-hoc networks, concluding that the extreme fairness emphasized causes harm in wireless multi-hop networks, and back up their conclusions with citations of observed 802.11 phenomenon in max-min fair implementations [45].

2.5.2 Source Rate Control for WSN

In this section we will primarily focus on unreliable, distributed transport layer protocols, as this is the flavor of congestion control supported by the current backpressure framework (though one could arguably implement reliability using fountain coding or other mechanisms). Limited mention of transport layer protocols outside this regiment will be discussed here, but alternatives deserve substantial consideration for the application-specific tradeoffs that result. Further, we note that some many protocols described in Section 2.6.1 employ rate limiting techniques, but unlike the work in this section are not designed for continuous flow-centric rate control.

The COngestion Detection and Avoidance (CODA) control scheme, published by Wan et al. [136], is one of the earliest distributed rate control protocols. The authors cite periodic event detection and network collapse as a common barrier to event-triggered large scale wireless sensor network deployments. Nodes detect congestion through a combination of channel loading estimation (carrier-sense probe based) and forwarding queue backlog. The emphasis of CODA is to mitigate the hotspots that occur in both
channel loading and queue backlog through a two-pronged approach of open-loop hop-by-hop backpressure (not to be confused with the term we use to refer to the stochastic network optimization framework) and closed loop source rate regulation. Wan et al. argue that these two approaches target different congestion scenarios, providing a complete solution for general congestion avoidance and application utility maximization.

The authors argue that their per-hop backpressure signaling handles queue backlog hot spots for short-duration events, but is not well suited to long-term source rate control. When rate control is necessary over a longer horizon, it becomes the task of a multi-source sink-centric token streaming scheme to limit sources and enforce fairness. Any source can request that the sink begin enforcing this sink-centric token streaming, as a result of the source observing a disproportionally low available rate. There is no pre-defined utility function to which any node is ascribing, the hysteresis threshold is based upon some fraction of the maximum theoretical channel capacity. We note that CODA was implemented atop the Directed Diffusion routing protocol [54], and was not capable of providing congestion feedback to the routing layer.

The hop-by-hop flow control of CODA is leveraged in Fusion, a subsequent proposal by Hull et al. [53] for network congestion avoidance in WSN. In total, the authors explore the impacts of three techniques for congestion avoidance in WSN: hop-by-hop flow control, closed loop source rate limiting, and MAC prioritization. Their performance exploration is carried out over a 55-node indoor testbed, Fusion incorporates all three techniques. Like CODA, the hop-by-hop flow control works to minimize packet drops caused by forwarding queue overflow. We note that this goal is very similar (though with different reason) to the Pull Collection Protocol (PCP) of Wachs et al. [135]. In PCP, the packet losses
were minimized in order to reduce debug energy requirements. The source rate control in Fusion more closely resembles the de-centralized approach of Woo and Culler [144]. Third, Fusion employs MAC layer prioritization for substantially backlogged nodes. When the hop-by-hop flow control and MAC layer prioritization are used in collaboration (Fusion) we note that the network stack is beginning to look quite a lot like an intuition-derived backpressure framework from stochastic network optimization, described in Section 2.4.8. There are substantial missing components, however, which make Fusion non throughput-optimal from a theoretical standpoint. Further, it is important to note that the metric for fairness used in Fusion is not max-min or proportional fair, but instead is borrowed from wired literature [57] (equal to the ratio of the square of the sum of rates to N times the sum of squared rates).

Finally, we note that the MAC prioritization employed by Fusion is a simple two-priority implementation, advocated by [1]. If a node’s forwarding queue exceeds a congestion threshold, the MAC layer pre-transmission random back off interval is reduced to $\frac{1}{4}$ that of low congested nodes. We will discuss a similar two-prioritization MAC we developed on a backpressure framework in Section 2.5.3. Unfortunately, Hull et al. [53] do not provide experimental results with only the MAC layer prioritization disabled while maintaining per-hop flow control and source rate control. It is therefore not possible to determine what if any benefits were a result of this MAC prioritization modification.

Recognizing the additive-increase multiplicative decrease (AIMD) wireless sensor network exploration by Woo and Culler [144], Rangwala et al. propose a more sophisticated implementation dubbed the Interference-Aware Fair Rate Control (IFRC) protocol [113].
Like Fusion, queue lengths are used to gage congestion onset, but unlike CODA or Fusion, IFRC specifically targets max-min fair source rate allocations (arguably a more useful fairness metric for WSN than the metric used in CODA [136]). Further, IFRC leverages explicit capacity knowledge and novel mechanisms to precisely detect the active flows interfering at bottleneck nodes. This explicit knowledge and more sophisticated information-sharing improve upon the work of Woo and Culler [144]. Experimentation on a 40 mote testbed is carried out by modifying MultiHopLQI to pin the routing after a reasonable tree is formed. The experimental results therefore do not reflect the capability of IFRC to support multi-path source rate control.

Noting the convergence time required (as long as 300 seconds) for IFRC, and the common queue occupancy of 8-10 packets, Sridharan and Krishnamachari [120] look to explicitly model the wireless capacity in order to avoid the use of queue occupancy to indicate congestion. The Wireless Rate Control Protocol (WRCP) operates on a model for wireless sensor network capacity that is similar in nature to the interference models of Hull et al. [53] and Rangwala et al. [113]. A key differentiator, however, is that WRCP leverages knowledge of reasonable per-receiver capacity, allowing it to divide this capacity uniformly amongst the competing flows. This results in very rapid rate adaptation and small queue sizes. Further, the authors implement bootstrapping mechanisms to allow rapid learning and adaptation upon inclusion of new network flows. Both IFRC and WRCP work to maintain max-min fair allocations, and we note that the capacity achieved in equivalent 40 node testbed evaluations yield little performance differentiation in this metric [120]. This is likely because both protocols experience the same bottleneck node,
a feature that would require dynamic or multi-path routing to alleviate, which is not presented in these publications.

While the works discussed so far were un-reliable in their delivery, we will now briefly discuss a recent work on reliable source rate control. In a paper describing the Rate-Controlled Reliable Transport (RCRT) protocol [104], Paek and Govindan emphasized rate control with reliable delivery. They focused these features within the sink node, as it has best visibility into the collection tree over which congestion may be occurring.

Paek and Govindan note that losses in many of the high sampling rate applications are particularly hurtful to application performance. In the loss of data from one sensor node at a particular sampling period, one may have to discard arriving data from multiple additional nodes due to it’s lack of usability. Further, a substantial source of losses can be attributed to network congestion and queue drops. It therefore makes sense to pair the reliability and congestion control features, as is proposed by RCRT. Additionally, the authors emphasize multi-stream collection, in which the sensor network is not transporting only a single data type. The authors implemented the sink-side reliability and rate control functionality in a PC-class platform, having greater resources and ease of code modification.

Another important feature of RCRT is its sink-centric support for diverse utility functions, requiring no modification to mote code in order to modify the resource determining per node utility function. This makes RCRT more flexible than many alternative rate control protocols such as IFRC [113] (supporting fair and weighted fair allocations), and WRCP [120] (providing lexicographical max-min fair allocations).
2.5.3 Backpressure-Based Rate Control Mechanisms

We previously investigated the performance of source rate utility optimization over static routing trees using system translations of backpressure theory at the MAC and protocol layers [121]. We termed the pure backpressure (PB) MAC one which chose to forward over the CSMA radio myopically, such that each node chose independently to forward the head of line packet when the backpressure link weight is positive. In 40 mote experiments on Tutornet, targeting lexicographic max-min rate fairness, we found that this simple PB MAC performs equivalently to more advanced mechanisms requiring neighborhood coordination.

Additionally, we found that the selection of $V$ in a real system becomes highly dependent upon source node count and topology, particularly for source rate utility functions having high insensitivity to $V$. Lexicographical max-min fair rates are achieved, for example, through the following utility function: $Utility_i(t) = \lim_{\alpha \to \infty} \frac{V}{(1-\alpha) \cdot R_i(t)^{1-\alpha}(t)}$ where $R_i(t)$ is the admission volume by user $i$ in timeslot $t$. The source rate admission decisions, obtained through the Lyapunov drift framework, dictate that the source rate be set as $R_i(t) = \sqrt[\alpha]{\frac{V}{Q_i(t)}}$ with $Q_i(t)$ the queue backlog of node $i$ in timeslot $t$. For topologies with few active sources, a small $V$ is suitable and results in small queue utilization (which impacts convergence time, network storage and delay). For topologies with all sources active, however, $V$ must be vastly enlarged to provide similar optimality. This work indicates that in backpressure stacks optimizing source rate utilities, $V$ parameter adaptation is likely necessary.
2.6 Notable System Protocols

2.6.1 Bulk Data Transfers

Bulk data protocols span many (if not all) layers of the network stack. We therefore discuss them here, as this is an important application space with substantial recent contributions. The goal of such bulk transfer protocols is either the dissemination of large data blocks (e.g. mote re-programming) or collection of sensed data (e.g. selective data retrieval as in structural monitoring in Wisden [103], volcano monitoring by FETCH and LANCE [140, 141, 142], and traffic monitoring [11] or un-filtered bulk transfer as in the golden gate deployment [68]). Further, the protocols may elect to be end-to-end reliable in their transfers or best-effort. We previously highlighted a number of these bulk data transfer mechanisms that were deployed in real systems, we refer the reader back to Section 2.1 for discussion of Wisden, FETCH, LANCE, and STRAW.

In design of the Trickle protocol [77], Levis et al. design and demonstrate a “polite gossip” algorithm to propagate incremental software updates to networks of very large scale with low energy overhead. The target application is broadcast in nature, and therefore somewhat dis-similar to the collection emphasis of our work. By carefully listening for and reacting to (i.e., employing a gossip-based rate control) meta-data summaries that are broadcast by nodes in the sensor network, Trickle achieves a message scaling of $O(\log n)$ with node density for any fixed messaging rate. Levis et al. integrated Trickle into Maté [76], a tiny bytecode interpreter for TinyOS Sensor Networks. Trickle allows the authors to update each of a set of small code routines run by Maté. In simulation and empirical evaluation of Trickle integrated into Maté, Levis et al. verify the very low
overhead of Trickle for networks of scale. The demonstration is somewhat specific to extremely small updates, as each Maté routine has been engineered to fit within a single TinyOS packet. Building on Trickle to support large data objects, Hui and Culler develop Deluge [52]. Primarily, Deluge represents performance improvements to Trickle’s implementation in supporting lower latency and greater update capacity while simultaneously keeping energy consumption minimized.

While the techniques employed in Trickle were very well suited for the application at hand, they were not designed for or applicable to data collection. To this end, and specifically citing the clear trend of time-synchronized bulk transfers in real world deployments, Kim et al. propose Flush [67]. Key properties of Flush include the use of end-to-end acknowledgments, snooping mechanisms for greater control information awareness and per-hop flow rate control while leveraging the simplifying assumption of no inter-flow interference (rationalized by the TDMA mechanisms of real world deployments).

The operation of Flush is somewhat unique, so we will discuss it here. In order to request a data transfer, the sink sends a request to the specific source of interest. This message is routed by an underlying (existing) delivery protocol. Once received by the source, a four phase process is initiated that includes topology query, data transfer, acknowledgment and integrity check. Flush does not specify the forward or reverse routing, but assumes the service is available on the sensor devices. During the topology probe phase the source determines the hop count to the sink. This information is used to tune packet transmission rate, to prevent intra-flow collisions in the multi-hop network. If the node is sufficiently deep, pipelining (spatial re-use) mitigates this backoff value. Note that this technique was employed by Kim et al. in their golden gate deployment [68].
Once the topology estimates have occurred (hop count, network capacity estimate), the source begins data transfer. Actively during transfer, each hop in the route employs dynamic rate estimation. The sink actively keeps a scoreboard of received data, so that it may recognize when the transfer is complete. At this point the sink sends Negative Acknowledgements (NACKs) to the source for packet ids that were lost in transit. Finally, once all portions of the data block are received, an integrity check of the data is carried out. If failed, the sink requests a fresh transfer of the entire data block. The authors do not pipeline the NACK packet transmissions, arguing that the end-to-end loss rate is sufficiently low that the added complexity is un-warranted. They cite a 3.9% loss rate in a 48-hop experiment, resulting in only two extra RTT of delay for a 760 packet data block request.

The rate control mechanism of Flush is also worth discussion. Using a channel access pipelining model similar to the Receiver Capacity Model [120], the authors describe an explicit rate control mechanism leveraging packet snooping and gossip based parameter dissemination. By determining the clearance time for a packet to be forwarded outside of interference range, then disseminating this information up and down the route, the source is capable of computing the maximum transfer rate that will avoid intra-flow interference (and therefore congestive collapse).

Finally, we note that Flush experiments were run on top of MintRoute [145] for forward-routing of data blocks to the sink. Kim et al. believe that Flush should perform well on any quasi steady-state routing algorithm, but worry that the rate estimation technique may not converge under routing protocols that support dynamic or multi-path routing. The data block requests and negative acknowledgements are broadcast through
the TinyOS flooding protocol Bcast. Kim et al. find that Flush is scalable, yielding fully 1/3 of the one-hop capacity in experiments on a 48-hop wireless network.

### 2.6.2 Protocols Employing Notions of Backpressure

There have been a number of publications over the last decade which leverage queue congestion information in order to make educated routing or forwarding choices.

In recent systems literature, The Volcano Routing Scheme (VRS), described by Ganjali and McKeown [36] uses potential routing to funnel packets to their destination over a wireless multi-hop topology. Node potentials are a hybrid of hop counts and packet backlogs, so as to describe the total congestion and hop count to the sink represented by the associated node. The potential routing approach of VRS is not proven to be throughput optimal, as is the case for Lyapunov Drift based algorithms. The potential term as defined in [36] differs from hop count minimization Lyapunov drift optimizations in that it has no link rate multiplier.

The authors of Horizon [112] demonstrate route selection and load balancing over an 802.11 mesh network. Their work diverges in key manners from the formal theoretical works mentioned earlier [97, 91, 37], in that they operate on a fixed and hand generated routes, presumed to be generated using an external routing protocol. The load balancing mechanisms therefore do not live up to the potential of true backpressure routing. Horizon also lacks a complete stack which includes routing, resource utilization, and source optimization, as we are working towards.

In an effort to address MAC implications as well as route load balancing, Warrier et al. [139] implement MAC backoff window prioritization as a function of differential queue
size, underneath back-pressure-motivated route selection similar to Horizon [112]. Again, an external routing algorithm is assumed to provide the multi-path routing alternatives.
Chapter 3

The Backpressure Collection Protocol

This chapter describes work that was presented in part in [88]. Section 3.5.5 gives experimental work by Paul Martin, who evaluated the Backpressure Collection Protocol in the context of localization.

3.1 Introduction

As wireless sensor networks mature from concepts and simple demonstrations to real-world deployments, there has been a push to identify and develop key networking building blocks in a more organized and coherent fashion. One such fundamental building block that has been identified at the network layer is Collection, which allows for data from multiple sources to be delivered to one or more common sinks. State-of-the-art implemented protocols for collection are based on quasi-static minimum cost trees with suitably defined link metrics [38]. Due to the limited radio link rates, high density of deployment, and multi-hop operation, bandwidth is a scarce resource in wireless sensor networks, and recent studies such as [11] have suggested that it is essential to improve
collection throughput as much as possible. Additionally, collection capabilities in real systems must be extremely robust to external interference, requiring routing responsiveness to sudden link fluctuations. Finally, new collection scenarios as seen in participatory sensing [20] demand responsive routing to sink dynamics even while maintaining substantial collection rates.

We explore in this work an exciting alternative approach — *dynamic backpressure routing* — whose per-packet next-hop route computations allow for greater responsiveness to link variation, queue hot-spots, and node mobility; this substantially enhances the throughput efficiency of collection. In this paper, backpressure routing refers to techniques grounded in stochastic network optimization [37, 78, 94, 91, 96, 97, 128] referred to as *Utility Optimal Lyapunov Networking* algorithms in recent work by Neely [96]\(^1\).

The crux of this approach lies in the generation of queue backlog gradients that decrease towards the sink, where these queue backlogs encode certain utility and penalty information. Using information about queue backlogs and link states, nodes can make source rate, packet routing and forwarding decisions without the notion of end-to-end routes. In theory, backpressure mechanisms promise throughput-optimal performance and elegant cross-layer solutions for integrating medium access, routing, and rate control.

Despite the theoretical promise of these dynamic backpressure techniques, to date they have not been implemented in practice at the routing layer\(^2\) due to several challenges. First, if the link weights are not carefully defined, backpressure routing can suffer from either excessively high hop-counts or, at the other extreme, over-emphasize low hop count.

---

\(^1\)In particular, we differentiate our work here from other heuristic queue-congestion aware load-balancing mechanisms, sometimes also referred to as backpressure mechanisms.

\(^2\)There have been several implementations of backpressure ideas at the MAC and transport layers, as we shall discuss when presenting related work in section 3.6.
counts, resulting in wasted transmissions and link-layer packet losses. Second, due to large queue sizes that must be maintained to provide a gradient for data flow, backpressure routing can suffer from inordinately large delays. Third, queues grow in size with distance from the sink which is a problem in large-scale deployments due to maximum queue size limitations in resource-constrained devices.

In this work, we take the first steps towards addressing these problems in order to allow backpressure routing to realize its promise in practical environments. We present the Backpressure Collection Protocol (BCP), a low-overhead dynamic backpressure routing protocol at the network layer implemented on TinyOS 2.x, a widely used wireless sensor network operating system and protocol stack. We evaluate it in real experiments on a 40-mote testbed, where we compare BCP’s performance with the Collection Tree Protocol (CTP) [38], a state of the art routing protocol distributed with TinyOS 2.x. Within relatively static networks having predicable topology and interference, we find that BCP performs competitively with CTP. The queue stability of BCP allows it to outperform CTP in terms of the max-min rate by more than 60%, and BCP’s ETX minimization reduces average packet transmissions by more than 30% versus CTP in low traffic tests.

In more adverse environments, such as those with unpredictable and severe external interference, we show BCP adapts quickly to link fluctuations, providing excellent packet delivery ratios and low average ETX. We show that by using LIFO queueing instead of FIFO, the delays associated with backpressure routing can be reduced dramatically, by more than 98% at low data rates and by 75% at high data rates, without appreciably affecting the achievable goodput. We introduce a novel concept of floating queues into the
backpressure framework, allowing for scalability in network size and load while maintaining throughput-utility performance and fixed sized data queues. Finally, we demonstrate excellent performance in participatory sensing settings, in which high sink mobility and multi-sink capability is desired.

In section 3.2 we give a more detailed description of backpressure routing. Section 3.3 discusses the challenges to practical systems implementation, and BCP’s novel solutions that address these challenges. The software design of BCP is described in section 3.4, and section 3.5 gives experimental evaluation. In section 3.6 we provide a brief survey of related systems and theory work. Finally, we conclude and discuss extensions in section 3.7.

### 3.2 Backpressure Explained

![Figure 3.1: An intuitive example of backpressure routing on a four-node line network with FIFO queueing service. Three packets (in black) are injected at nodes 1 and 2 at time B, intended for the destination sink S.](image)

Unlike traditional routing mechanisms for wired and wireless networks, backpressure routing does not perform any explicit path computation from source to destination. Instead, the routing and forwarding decision is made independently for each packet by computing for each outgoing link a backpressure weight that is a function of localized
queue and link state information. Before presenting the detail of the Backpressure Collection Protocol (BCP), we present a simplified introduction to the basic concepts and theory behind backpressure routing.

### 3.2.1 Routing as a Stochastic Optimization Problem

We first present a rigorous definition for a stable network. Let $Q_i(t)$ being the backlog of the queue at node $i$ during time slot $t$. We call a network of queue backlogs strongly stable if:

$$
\limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[Q_i(\tau)] < \infty \text{ for all } i \tag{3.1}
$$

Additionally, let $f(\bar{x}(t))$ be the penalty resulting from control decisions in time slot $t$ (e.g. $\bar{x}(t)$ being the routing and forwarding decisions between queues and $f()$ the power consumed by these decisions) with $f$ non-negative, continuous, convex and entry-wise non-decreasing ($f(\bar{x}) \leq f(\bar{y})$ whenever $\bar{x} \leq \bar{y}$ entry-wise). Let $f(\bar{x})$ be the value of the penalty function operating on the long term average value of the $\bar{x}(t)$ vector. We can then formulate a stochastic network optimization problem in which control decisions minimize the penalty function while maintaining strongly stable queues:

**Minimize:** $f(\bar{x}) \tag{3.2}$

**Subject to:** Strongly Stable
Specifically the problem of routing can be formulated in the above form by assuming that $f(\bar{x})$ is some cost metric for routing. The solution for Equation (3.2) using the Utility Optimal Lyapunov Networking framework [37, 100, 91, 97] can be shown to be control decisions resulting in a backpressure routing policy. In this routing policy each node calculates the following weight per outgoing link in every time-slot:

$$w_{i,j} = (\Delta Q_{i,j} - V \cdot \theta_{i,j}) \cdot R_{i,j}$$  \hspace{1cm} (3.3)

Here, $\Delta Q_{i,j} = Q_i - Q_j$ is the queue differential (backpressure), with $Q_i$ and $Q_j$ representing the backlog of nodes $i$ and $j$ respectively, $R_{i,j}$ is the estimated link rate, and $\theta_{i,j}$ is a link usage penalty that depends upon the particulars of the utility and penalty functions of (3.2). The $V$ parameter is a constant trades system queue occupancy for penalty minimization. In the theoretical framework, the optimal solution requires centralized scheduling of the set of non-interfering links at each time that maximize the sum of these weights.

A key advantage of formulating the problem of routing, as shown above, is that since the routing policy is striving to minimize the penalty $f(\bar{x})$, it will intuitively minimize looping of packets in the network, since any loops result in an unnecessary increase of the routing penalty $f(\bar{x})$. The validation of this hypothesis will be presented in our empirical evaluation in section 3.5.

In our decentralized approximation to the optimal backpressure routing policy, node $i$ computes the backpressure weight $w_{i,j}$ for all its neighbors, and uses it as the basis for
making independent routing (who to try and send the packet to) and forwarding (whether to transmit the packet) decisions as follows: **Routing decision:** Node $i$ identifies the link $(i,j^*)$ with the highest value of the backpressure weight as the next hop for the packet. **Forwarding decision:** if $w_{i,j^*} > 0$, the packet is forwarded (i.e. sent to the link layer for transmission to the designated neighbor), else the packet is held until the metric is recomputed.

### 3.2.2 A Simple Example

In describing this simple routing and forwarding mechanism to colleagues unfamiliar with backpressure techniques, we have found that a common initial reaction is surprise that this simple forwarding strategy that has neither an explicit path computation nor an explicit reference to the destination, should work at all.

We will first illustrate the functioning of backpressure routing with a very simple example. Figure 3.1 shows a network of queues with four nodes labelled 3, 2, 1, and S (for sink) respectively. For simplicity, assume that each link has $\theta_{i,j} = 1, V = 1, \overline{R}_{i,j} = 1$.

In steady state, as shown in step A, there is a natural queue backpressure gradient sloping downwards to the sink \(^3\). Each node has just one packet more than its neighbor to the right but is unable to forward because it does not strictly exceed the threshold of $(V * \theta_{i,j} = 1)$. The injection of new packets into nodes 1 and 2, shown in step B, causes the thresholds to be exceeded. Node 1 then starts sending packets to the sink, while node 2 initially forwards a packet backwards to node 3 (after step B), then halts (after step

---

\(^3\)Backpressure routing requires a gradient to exist before packets can begin to be forwarded, resulting in a small startup time, and the possible sacrifice of a small number of “trapped” packets. Both of these are negligible concerns for even moderately long flows.
C), then reverses to start sending packets to node 1 as that node’s packets are drained out by the sink. Eventually six packets (corresponding to the number of new arrivals) are sent to the destination, and the system returns to the steady state gradient.

### 3.3 Novel Contributions of BCP

Figure 3.2: A three node network is given in (i), links are labeled with both rate and expected transmission count per packet. Bold links in (ii) through (iv) indicate links selected for packet forwarding. Weights are calculated using Equation (3.5) with \(V = 1\).

There are three key challenges in translating backpressure routing from theory to practice. The first challenge is to choose an appropriate penalty function \(f(\bar{x})\) in equation (3.2) to provide efficient performance over a real-world wireless network with lossy links. The second challenge is that traditional backpressure approaches suffer from high end-to-end packet delays because they inherently rely on having large queue-backlogs to provide throughput-optimality. A related third challenge is that large queue sizes required by traditional backpressure approaches are difficult to support on resource-constrained wireless sensor devices.

We now discuss each of these challenges in more detail and present our novel contributions in the design of the Backpressure Collection protocol that address each of them.
3.3.1 ETX Minimization

The expected number of transmissions (ETX) required to successfully transmit a packet from a sender to a receiver is a commonly used metric for routing in wireless multi-hop networks with lossy links [26, 145, 16, 38]. To incorporate ETX minimization into the backpressure framework, we use the following penalty function in the optimization problem (3.2):

$$f(\vec{x}) = \sum_i \sum_{j \in \mathcal{N}_i} x_{i,j}(t) \cdot \text{ETX}_{i,j}$$  

(3.4)

Where $\mathcal{N}_i$ is the set of neighbors of node $i$, $\text{ETX}_{i,j}$ is the link ETX estimate, and $x_{ij}(t)$ is the forwarded packet count over link $i \to j$. Note that $f(\vec{x})$ satisfies the penalty function properties of problem (3.2), and yields a backpressure weight $w_{i,j}$ calculated by a node $i$ for a given neighbor $j$ which is the following:

$$w_{i,j} = (\Delta Q_{i,j} - V \cdot \text{ETX}_{i,j}) \cdot \mathcal{R}_{i,j}$$  

(3.5)

By including ETX as a link penalty, BCP works to minimize ETX when possible, while maintaining strongly stable queues. For a more thorough understanding of BCP weight calculations and how traffic conditions affect routing dynamics, we’ll next consider the small network of Figure 3.2 (i). As was observed in Figure 3.1, the forwarding penalties exceed queue differentials, causing the network to stall while waiting for additional
admissions. When admissions do occur at the source, shown in (ii), the weight is greatest between the source and node B. Note that node B is on the path with lowest source to sink ETX. Packets forwarded to node B then trigger the weight between node B and the sink to become positive (iii), resulting in delivery to the sink. Should periodic source arrivals continue without seriously stressing the network capacity, a flow of packets will cascade from the source to node B and then on to the sink. In the event of a sudden load increase that causes node B’s queue to back up, such as seen in (iv), the source to sink link’s higher capacity influences the weight maximization and the network reacts to the loading threat (hot spot) by forwarding directly to the sink.

### 3.3.2 Delay Reduction using LIFO

![Figure 3.3: The four-node network of Figure 3.1, now with LIFO service priority. New additions to the queues flow over the existing gradient to the sink.](image)

High source to sink delays are a well established problem in backpressure systems, resulting in significant recent theoretical focus [18, 50, 93, 96, 149]. Under a FIFO service priority, data reaches the sink only when it is pushed through the chain of queues toward the sink. Counter-intuitively, the average source to sink delay in backpressure algorithms under FIFO service priority grows with decreased loading for low loading [18]. Halving the admission rate across the network can double the per-source average packet delivery delay. This puts traditional FIFO based backpressure algorithms at a severe delay disadvantage when compared to tree routing alternatives.
Our novel delay solution is motivated by imagining water cascading down the queue backpressure gradient that is built up in steady state. Intuitively, this way, instead of packets having to make their way through all the queues, new packet arrivals can be rapidly sped to their destination. This can be achieved by using a last-in-first-out (LIFO) queueing discipline. We illustrate this in figure 3.3. Note that by the time the system returns to the minimum queue backlog state in step J, all new arrivals have been delivered to the sink. None of our admissions become trapped within the queues, waiting to be pushed toward the sink by future arrivals.

In the Appendix, we prove that for queue i with minimum recurrent backlog $b_{i}^{\text{min}}$ and arrival rate $\lambda_i$, the average delay of a packet serviced by LIFO priority ($W_{i}^{\text{LIFO}}$) is unboundedly lower than that achieved by FIFO priority ($W_{i}^{\text{FIFO}}$):

$$W_{i}^{\text{FIFO}} = W_{i}^{\text{LIFO}} + \frac{b_{i}^{\text{min}}}{\lambda_i}$$  \hspace{1cm} (3.6)

Note that some packets may be trapped within the LIFO indefinitely; however this approach drastically reduces the serviced packet delay. We show empirically in section 3.5 that this innovative use of LIFO in backpressure routing reduces the average packet delay for packets reaching the sink by at least two orders of magnitude at low data rates, when compared with FIFO queuing.
3.3.3 Scalability

Another substantial challenge faced by systems implementations of backpressure techniques is scalability. It can be clearly seen in Figure 3.2 that the minimum queue size grows with each hop from the sink (by at least the corresponding ETX $\geq 1$). Given the extremely limited queue availability in resource-constrained wireless sensor nodes, therefore, nodes beyond a certain number of hops end up with saturated queues, resulting in improper routing and forwarding decisions. We will demonstrate this empirically in section 3.5.2.2.

Recent theoretical work on queue stability [50] in the context of backpressure schemes shows that the tail distribution of queue backlogs shrinks exponentially beyond some distance from the mean value determined by the queue gradient at steady state. We have also verified empirically that the queue backlog distributions tend to be concentrated around their long term average values, suggesting that much of the queue is not used in accommodating traffic bursts, but instead only incurs delay while consuming system
resources. We leverage this fact to generate floating queues: a scalable solution that does not break the optimization framework. Analytical performance guarantees of the floating queue are omitted here due to their lengthy derivation and are the subject of future publication.

Consider the trapped white packets in Figure 3.3. Because the queues can never drop below their minimum backlogs shown in (A) and due to LIFO service priority, these white packets will never be serviced. Our floating queues discard these white packets and add them to a virtual queue, which then lies beneath the data queue. We carefully balance queue arrivals and departures (using null packets if needed) so as to preserve the overall stochastic queue dynamics, and backpressure weights are computed on the combined virtual plus data backlogs. A floating queue is illustrated in figure 3.4. We show empirically in section 3.5.2.2 that without these floating queues, backpressure routing does not scale to large-diameter networks, and that the overhead of null packets is negligible.

### 3.4 BCP Implementation

We have developed the Backpressure Collection Protocol (BCP), the first ever real-system implementation of a dynamic backpressure routing mechanism. BCP is implemented on TinyOS 2.x, and has been tested on the IEEE 802.15.4-based Tmote Sky platform\(^4\). BCP’s code footprint is about 23 KB including our test application, versus CTP’s 27 KB.

\(^4\)BCP source code is publicly available online at http://anrg.usc.edu/~scott/
3.4.1 Routing and Forwarding

The routing and forwarding algorithm for BCP is simple. When the forwarding queue is non-empty, weights are computed for every neighbor using Equation (3.5). If all weights are less than or equal to zero, there exists no good neighbor option and the node waits for a back-off period $\tau$, then re-computes weights. Eventually, arrivals or neighbor queue backlog values will cause a neighbor’s weight to become positive. Upon detecting one or more positive neighbor weights, the node forwards the head of line packet to the neighbor having greatest positive weight, then repeats the weight computation process.

3.4.2 Weight Recalculation

The weight recalculation parameter $\tau$, which determines the time for which a packet that is not forwarded is withheld before the metric is recalculated, provides a tradeoff between throughput/delay performance and processor loading. We use $\tau = 50ms$ for our experiments, resulting in weight re-computation (section 3.4.1) 20 times a second in the event that no neighbor has a positive weight. We can use a larger $\tau$ in case weight re-computation keeps the CPU too busy for other tasks on the node.

3.4.3 Link Metric Estimation

Link metric estimation for transmissions $\overline{ETX}_{i\rightarrow j}$ and rate $\overline{R}_{i\rightarrow j}$ are carried out in an online fashion by each node $i$ for each of its neighbors based on local time stamps of its unicast data transmission attempts and corresponding received acknowledgments. The metrics are updated using exponentially weighted moving averages (BCP uses a simple stop-and-wait ARQ with a maximum of 5 retransmission attempts on a link before weights
are recomputed). For both our EWMA estimates of ETX and link rate, we use a weight of 0.9 for the previous estimate. We show in section 3.5 that the responsiveness of BCP to external interference is quite good with this parameter setting.

3.4.4 Disseminating Local Queue Backlog

Each data packet includes a packet header field for disseminating the local queue backlog. The header fields for BCP are identical to CTP with two key exceptions. One is that the ETX field for CTP is used by BCP to broadcast the local queue backlog, and the other is that one of the reserved bits is used to flag null packets (described in the next subsection). All the nodes within reception range of the transmitter receive and process the BCP packet header through the snoop interface. We note that snooping mechanisms have been employed in alternative WSN protocols (e.g. Wisden’s cache recovery [103], MintRoute link estimation [145], Flush control message passing [67]) and some 802.11 protocols (e.g. COPE [66]). In order to reduce the potential for processor overloading, a small 5-packet FIFO is attached to the snoop interface, allowing for snoop message drops in the event of microprocessor overloading and quick returns from radio snoop events. Experimentally, this has proven necessary to prevent processor overload due to packet snooping. Using this snooping mechanism, BCP incurs no additional overhead in terms of separate broadcast control packets for either link estimation or for exchanging queue status. Additionally, these snooped backlog updates provide frequent notification of neighbor congestion, as is required by backpressure techniques.
3.4.5 Floating Queue Implementation

Our LIFO floating queue is implemented through the introduction of a virtual queue, which stores no real data and requires only an integer size. When data arrives to the forwarding queue and finds it full, the oldest data packet is discarded from the data queue and the virtual queue is incremented. When servicing the forwarding queue, if the data queue is found to be empty we instead generate and forward a null packet, reducing the size of the local virtual queue. Null packets are filtered by the sink and statistics are kept on their arrival rate. Local backpressure is computed by summing both the forwarding queue size and this virtual queue.

3.5 Experimental Results

3.5.1 Experimental Methodology

We perform our evaluation experiments on motes labeled 1-40 in Tutornet, an indoor wireless sensor network testbed consisting of IEEE 802.15.4-based Tmote Sky devices. A transmit power of -18 dBm is used for all experiments, and packet inter-arrival times are exponential, providing *poisson* traffic. Packet sizes were 34 bytes for both BCP and CTP, as both protocols require an 8 Byte data packet header in addition to the CC2420 header (12B) and payload (14B).

While there have been previous theoretical/simulation-based proposals for use of multi-path routing in wireless sensor networks [35], nearly all implemented routing protocols for collection in wireless sensor networks have been based on minimum cost trees, including the state-of-the-art collection tree protocol (CTP) [38], which we have used
as a baseline in our evaluation. CTP has been thoroughly validated and released as a routing protocol for TinyOS 2.x, and has been used for comparison purposes in a number of recent works [135, 24, 31, 30, 23]. In our experimentation, CTP uses the 4 bit link estimator (4bitle [31]).

A number of BCP variants will be evaluated in order to demonstrate the improvements garnered from floating LIFO queues. All references to BCP imply the core BCP implementation having floating LIFO queues enabled. In 3.5.2, we first run experiments within a simple collection scenario, where external interference is minimized and topology is held constant. After demonstrating competitive performance with CTP in these less arduous environments, we move on to settings with strong external interference (3.5.3) and finally high sink mobility (3.5.4).

### 3.5.2 Static Network Tests

The following static scenarios all run on 802.15.4 channel 26, as this channel does not experience external 802.11 interference within the testbed environment. All tests collect data for 35 minutes, and 39 motes source traffic. For brevity, we will state only the per-source packet rate below, with the understanding that 1.0 packets per second indicates 39 sources are each active at this rate. The backpressure optimization parameter V was set to 2 as a result of early experimentation. We conservatively set \( \tau = 50 \text{ ms} \). Source rates vary from 0.25 to 1.66 packets per second.

---

5Due to space constraints, we do not provide a more detailed evaluation of the parameter settings for V and \( \tau \) in this paper; there may be potential for further improvement by careful parameter tuning. Our experiments, however, do show that the current settings are robust to dynamics in traffic, external interference and sink mobility.
3.5.2.1 Delay Performance

As discussed in section 3.3.2, delay in FIFO backpressure stacks actually grows with decreased loading, putting traditional backpressure at a severe disadvantage when compared to tree routing algorithms. Figure 3.5 provides the CDF of delivered packet delays for mote 4 (top) and mote 40 (bottom) in our 35 minute static network tests of CTP, BCP-FIFO and BCP-LIFO. Mote 4 is a single hop from the sink for all experiments, while Mote 40 is at the rear of the network and averages 5 hops from the sink in both CTP and BCP. Although still higher than the delay for CTP, the delay for delivered packets under LIFO service priority is two orders of magnitude lower than FIFO, for both motes 4 and 40. The system average delivered packet latency was 231 ms under LIFO, and 20,704 ms under FIFO. Experiments at 1.5 packets per second demonstrate a lesser delay improvement. Here, system average delivery delays under LIFO (FIFO) are 1,088ms (5,623ms). In all experiments, the percentage of non-delivered packets was indistinguishable for LIFO and FIFO service priorities (< 2% for 0.25 PPS, < 0.7% at 1.5 PPS). Undelivered packets are due to the learning time required within BCP. Whereas LIFO traps some packets sourced early in the flow, FIFO stalled when flows terminated, trapping the undelivered packets at the tail of the experiment.

Within the computer networks community, LIFO service priority is traditionally avoided for datapath queues. One key concern has been the reordering of packets in the network, which is much less common under static routing and a FIFO service priority. It is now known, however, that packet reordering is a challenge for multi-path routing algorithms, even under FIFO assumptions [106]. Reordering density ([107], RFC 5236
Figure 3.5: Source to sink delay CDF at 0.25 PPS for motes 4 and 40 under CTP, BCP-FIFO and BCP-LIFO.
Figure 3.6: The Reordering Density for BCP under FIFO (top) and LIFO (bottom) servicing priorities for 0.25, 1.0 and 1.5 packets per second per source. The quasi-static tree routing mechanisms of CTP resulted in greater than 99.9% in-order delivery for 0.25 and 1.0 PPS tests.
is a commonly accepted metric for analysis of reordering performance in a network. The packet transmission order is compared with delivery order and a pdf of observed reordering magnitude is generated. A reordering magnitude of zero indicates that the mth packet sourced was also the mth packet sinked, while a magnitude of one indicates that it is sinked (m − 1)th or (m + 1)th.

Figure 3.6 gives the reordering density for BCP under both FIFO and LIFO queue prioritization within our 35-minute static network tests at 0.25, 1.0 and 1.5 packets per second. The reordering density was also calculated for CTP at 0.25 and 1.0 packets per second (CTP was unstable beyond 1.0 packets per second) and more than 99.9% of all packets were delivered in order. At the lowest data rate of 0.25 packets per second per source, we find that BCP’s LIFO in order delivery rate (96.8%) is in fact greater than that seen under our FIFO test (92.7%). We attribute this to the enormous packet delay disparity between FIFO and LIFO; For most packets, the LIFOs allow for delivery to the sink before the source generates its next packet. This reordering trend reverses once queues become less stable. In Figure 3.6, at 1.5 packets per second, 3% of packets experienced reordering greater than 8 under BCP’s LIFO use, while the FIFO queues result in 2.2% of packets falling within the same reordering range. We therefore conclude that even when operating near the capacity region, the reordering penalty for LIFO use is small due to the natural packet reordering caused by multi-path routing.

### 3.5.2.2 Scalability

In section 3.3.3, we described the scalability challenge of backpressure stacks, and provided a description of our floating queue solution. In order to validate our solution, we ran
Figure 3.7: Comparison of BCP’s per mote goodput, time average queue sizes and source-to-sink average packet transmissions per packet per source (with 95% confidence interval). Tests are run with and without BCP’s floating queues disabled. The maximum data queue size is 11.
our 35 minute, 40 mote tests at 1 packet per second with BCP’s floating queues disabled. Figure 3.7 gives per mote goodput, time average queue size and average source-to-sink packet transmissions per packet per source. Note that without floating queues, motes furthest from the sink (highest ETX, which is correlated with mote ID loosely) reach their maximum queue occupancy and cease local admissions. Some motes (33-35, 39, 40) sink no packets at all, while others (36-38) successfully deliver only a few.

With floating queues enabled, backpressure information is no longer truncated by the size of the data queue. Instead, trapped LIFO data is discarded and an underlying virtual queue is incremented, allowing the sum of the data and virtual queues to represent backlog values greater than the size of the limited capacity data queue. This activity can be seen in the time average queue sizes in Figure 3.7, where some motes (35-40) have average queue size that exceeds the data queue capacity. We see that by using floating queues, BCP achieves greater than 98% delivery for all sources. Furthermore, the accurate backpressure information allows the ETX minimization to operate accurately, as true neighbor costs are reflected by advertised queue sizes throughout the network.

Floating queues do come at a potential cost, as it is possible for data queues to underflow, requiring service of the underlying virtual queue in order to maintain proper backpressure signaling. With the fixed data queue of 11 packets, the null packet delivery rate caused by floating queues was less than 0.2%. We found this very low null packet generation rate held for all experiments run on Tutomnet.
3.5.2.3 Goodput and Delivery Efficiency

Having addressed the primary barriers to system use, we next investigate the real-world performance of BCP. Figure 3.8 provides the goodput at the sink over various offered loads. At source rates in excess of one packet per second we begin to see packet losses over CTP resulting in a decrease in the minimum source rate. The cause is dominated by queue tail drops near the sink, indicating the need for source rate control. The BCP performance, however, demonstrates no significant losses until source rates exceed 1.66 packets per second per source, a more than 60% improvement in max-min rate. We attribute this to the queue-aware hotspot avoidance of BCP and its multi-path-like routing capabilities (though no explicit routes are employed).

![Figure 3.8: Goodput versus source rate in static network tests of BCP and CTP.](image)
While avoiding hotspots, BCP should also be minimizing packet transmissions, as this is one of the dual goals of our underlying stochastic optimization. Figure 3.9 provides per-source average transmissions to the sink for BCP and CTP at 0.25 packets per second and 1.0 packets per second per source. At 0.25 packets per second, the system average transmissions per packet for BCP (CTP) is 2.39 (2.65), while at 1.0 packets per second the average is 3.12 (2.99).

![Figure 3.9: Average source-to-sink transmission count per packet per source (with 95% confidence interval) for the static 1.0 PPS experiment. Flow sources are sorted by average transmission count for BCP.](image)

Having demonstrated that the performance of BCP is competitive with that of CTP, a tree routing protocol optimized for this static-network environment, we next move into the domain in which backpressure algorithms should intuitively excel.
3.5.3 External Interference

Due to the frequency sharing of 802.11 and 802.15.4 radios, and the severe disparity in transmit powers, 802.15.4 devices suffer greatly from external interference. We therefore ran a series of controlled external interference experiments in order to evaluate the external interference performance of BCP. Within Tutornet, mote channel 26 is reserved for low external interference tests, as this spectrum is shared only by 802.11 channel 14, which is unused in the building. For our external interference tests we operate two 802.11 radios (near nodes 25 and 33) on channel 14, transmitting UDP packets of size 890 bytes. The test begins with activation of source mote traffic at 0.25 packets per second per source for all 40 motes. After five minutes of settling time, we begin broadcasting packets from the 802.11 devices at a rate of 200 packets per second. During the interference phase, broadcasts are generated with a duty cycle of 10 seconds on, 20 seconds off. This periodic external interference continues for a total of 15 minutes, and we then conclude the experiment without external interference for a final five minutes.

In the top portion of Figure 3.10, we provide delivery ratios for each 30 second window of the experiment. The introduction of interference at 300 seconds is particularly clear in CTP performance, where delivery ratios fluctuate between 55% and 84% for fifteen minutes. Over the same test period, BCP’s delivery performance was markedly better: between 88% and 96% of packets reached the sink successfully. Inspection of tree-rebalancing messages in CTP indicate that the on/off behavior of the external interference caused frequent tree re-generation. Every such tree modification risks looping and misrouting behaviors, which can briefly result in queue discards. This can be seen in the
Figure 3.10: 30 second windowed average sourced packet delivery ratio (top) and system transmissions per packet (middle). Spectrum analyzer results are plotted at bottom for the interfering 802.11 channel 14 traffic.
bottom portion of Figure 3.10, where we plot system average transmissions per packet for each 30 second window of the experiment. The lack of end-to-end routing paths in backpressure protocols is a significant strength in this external interference test, allowing packet routing to respond to sudden external interference with fewer packet losses and better link selection.

3.5.4 Highly Mobile Sinks

There has been interest in the use of external mobile sinks within participatory sensing literature [20]. With the goal of evaluating BCP performance under such a scenario, we chose to emulate in experimentation the existence of an external mobile sink that repeatedly wandered through a portion of the Tutornet testbed. We selected a sequence of 17 sink motes, leading in order from several laboratories down a hallway to a stairwell. One mote at a time turns on its sink designation, as if an external sink had made contact and requested data. The sink designation moves every second, approximately the walking speed of a student, by cycling through a sequence of motes with a 17 second period. All 40 sources operate at 0.25 packets per second during the 35 minute test.

It is important to note that CTP was not designed for highly mobile sinks, but prior system implementations for mobile sinks (such as Hyper [117]) assume much lower sink mobility than modeled here (minutes, not seconds). We believe that performance results for CTP will not be unreasonable as approximations for Hyper and other tree-centric sink mobility solutions. Table 3.1 gives the delivery ratio and average transmissions per packet for our sink mobility experiment for BCP and CTP. The delivery ratio and average transmission count for BCP improved when compared with the static network tests. The
(a) A 200 second window of sink time versus source mote for sinks 8, 18 and 26 running BCP (top) and CTP (bottom). Each Sink in BCP services communities of nodes having low ETX, while CTP attempts to rebuild trees that service the entire network.

(b) Circle radius represents the goodput received by the sink (row) originated by the source (column) for BCP (top) and CTP (bottom). Gradients in BCP direct packets to the most efficient regional sinks.

Figure 3.11: Comparison of performance of BCP and CTP under extreme sink mobility.
same cannot be said for the tree-based collection algorithm. To better understand the disparity, we next look into source-sink pairings in the mobile test.

<table>
<thead>
<tr>
<th>Mobility</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCP</td>
<td>CTP</td>
</tr>
<tr>
<td>Delivery Ratio</td>
<td>0.996 0.590</td>
</tr>
<tr>
<td>Average Tx/Packet</td>
<td>1.73 9.5</td>
</tr>
</tbody>
</table>

Table 3.1: Test results for highly mobile sink experiment at source rate 0.25 packets per second per source, provided alongside static network results from section 3.5.2.

In order to visualize packet deliveries over time, we plot sink timings per-source for sinks 8, 18 and 26 in Figure 3.11(a). We immediately note that once trees are generated by CTP (if successful) data from the entire network is rapidly routed to that new sink for the remaining sink duration (e.g. sink 8 at 725 seconds). This contrasts sharply with BCP, where motes draw regionally from their neighborhood (e.g. sink 8).

Plotting source to sink delivery goodputs, as in Figure 3.11(b), gives a system view of the packet handling under mobility for BCP and CTP. Looking at any particular source (column), under a tree routing algorithm (CTP at bottom) we observe a generally uniform traffic delivery rate to sinks in the network. Under BCP, we see that for many sources the sink delivery rate is highly uneven. These sources are forwarding the majority of their traffic to an intermittently available local sink, having low ETX distance. This results in very low average TX count per delivered packet.

3.5.5 Application Experiment

In order to explore the application-layer performance of BCP and CTP, experiments were run on Tutornet for the purpose of mobile node localization. The mobile mote...
beaconed at a rate of 5Hz, while static motes in Tutornet would measure RSSI and relay the reception strength metrics back to the sink for processing by a desktop computer. Windowing was employed to average RSSI reception measurements, and the Sequence-Based Localization (SBL) techniques of Yedavalli et al. [148] were applied. Packets generated by the listening mote (per overheard beacon message) were 36 Bytes in size.

Ten motes from Tutornet (indexes 1, 6, 12, 16, 30, 36, 45, 49, 27, and 40) were selected as overhearing and collection nodes, the power level for transmission of both the localization beacon and collection protocols was set to 0 dBm. All 56 motes were programmed to assist in data collection (either BCP or CTP). Motes were initialized and programmed with either BCP or CTP for data collection, and after allowing approximately one minute for network setup a student walked the path shown in Figure 3.12. Because BCP requires learning through data loss, the student had to walk the path twice in BCP experiments (otherwise loss ratios were so high as to provide no usable data). Once the BCP training phase had occurred, loss rates of 18% were typical for each lap of the student under BCP. By comparison, the CTP loss rate averaged 32%.

In Figure 3.13(a) we see the route estimate arrived at by SBL over the data provided by BCP. We note that the delay of delivered RSSI measurement was not a barrier to implementation over BCP. This implies that our substantial delay reduction techniques (credited to the LIFO implementation in BCP) was sufficient for this real-world application. The estimated mote location run atop BCP appears approximately reasonable, though clearly there are substantial errors. The location of the fixed listening motes within Tutornet posed an algorithmic challenge to accurate localization. This was because the student movement along the hallways was directly beneath (approximately) the
listening motes, which then could not provide accurate RSSI differentiation to the SBL algorithm.

In subsequent application execution over CTP, seen in Figure 3.13(b), we observe problems cropping up due to packet losses. As was discussed earlier, when CTP becomes congested the primary loss mechanism is queue tail drops near the sink. Therefore we see a disproportionate loss of RSSI measurements from the rear network nodes, effectively reducing the geographic diversity of our listening motes and severely hampering the localization algorithm. This highlights a feature hidden by the marginal difference in per-lap loss rates (18% for BCP, 32% for CTP). The near-sink loss rates were quite low for CTP, but this was achieved through a disproportionate loss in the rear network nodes.

In fairness, we would like to point out that the sampling rate of 5Hz was selected as a differentiating beacon rate between BCP and CTP. Though losses in BCP occur more uniformly across the network, which is advantageous in localization applications,
(a) The localization experiment run atop BCP after allowing BCP learning to complete (requiring one additional lap by the student).

(b) The localization experiment run atop CTP, routing tree construction requires no loss of data packets (and therefore no initial lap by the student).

Figure 3.13: Comparison of localization performance for a sample run. The rear-network originated losses to which CTP is prone (due to queue tail drops near the sink) are causing loss of RSSI measurements in the rear of the network, thereby hampering localization performance.
the network collapse is quite sharp with scaling of the beacon rate. The approximately 50\% enhanced capacity region of BCP, documented earlier, means that under a beacon rate of 3Hz the protocols perform quite competitively. The experiment presented here teases out the relative performance at the limit of CTP’s collection rate, but should not be taken to indicate broadly the relative capabilities of the very different routing protocols.

### 3.6 Related Work

The intellectual roots of dynamic backpressure routing for multi-hop wireless networks lie in the seminal work by Tassiulas and Ephremides [128]. They showed that using the product of queue differentials and link rates as link weights for a centralized maximum-weight-matching algorithm allows any traffic within the network’s capacity region to be scheduled stably. Recent work in Utility Optimal Lyapunov Networking [37, 78, 94, 91, 96, 97] has provided a theoretical framework for backpressure-based stochastic optimization that we have used in this work to derive the link-specific thresholds for BCP. Independently, Stolyar et al. have also developed a related backpressure-based stochastic optimization framework using Lagrange duality [18, 81, 126].

Researchers working on both optimization frameworks have lately worked to address delay reduction theoretically [18, 50, 93, 96, 149], but none stray from traditional FIFO assumptions. Our floating queue is similar in spirit to the virtual queues used in [50], but we require no knowledge of the steady-state optimal queue backlogs.

There has been work to translate backpressure scheduling and optimization into practical protocols for wireless networks. These efforts have been limited to the MAC and
Transport layers. In wGPD [4] and DiffQ [139], the queue differentials are used to change contention behavior at the MAC layer. In Horizon [112], load balancing decisions over multiple disjoint routing paths (generated separately by a link state routing protocol) take into account queue state information to enhance TCP performance. Transport layer backpressure-based rate utility optimization is demonstrated in both wGPD [4] and in the work by Sridharan et al. [121, 122]. None of these prior works have implemented dynamic backpressure routing at the network layer, which is the focus of BCP. Moreover, these prior works have not investigated the mitigation of delay in backpressure based protocols, nor scalability issues, key contributions of this work.

The routing weights computed in backpressure algorithms are related to a class of routing algorithms known as potential routing algorithms. With the goal of using queue congestion to make routing decisions, Ganjali and McKeown proposed Volcano [36], a potential routing algorithm that has some similarities to BCP. However, Volcano routing was evaluated only via idealized simulations and is not based on any theoretical framework; in particular, because it assumes lossless links, it effectively tries to minimize hop-count, which results in poor performance in real systems. Also, it does not provide any solutions for reducing queueing delay.

There have been proposals to use path or one hop queue backlogs in routing decisions for a number of protocols over the years, and modern protocols such as MIXIT [65], Horizon [112] and Arbutus [109] continue to see the strength of including congestion notification at the routing layer. These path-based congestion aware mechanisms are not derived from the Lyapunov Network Optimization framework, and are essentially heuristics that attempt to reduce queue drops by either re-computing routes or re-balancing...
rate allocations across multiple static paths when hot-spots occur. While they may improve throughput performance, we believe that such path-based heuristics will not be as responsive as BCP to dynamics that are disruptive to path construction and maintenance (such as sink mobility). We had hoped to do a direct comparison with one of these schemes to justify this claim, but were unable to do so at present due to lack of source code availability for the TinyOS platform.

3.7 Conclusion and Future Work

We have presented BCP, a collection protocol for sensor networks that is the first-ever implementation of dynamic backpressure routing in wireless networks. In BCP, we have implemented several novel techniques to make backpressure routing practical, such as ETX optimization and the use of floating LIFO queues. We have shown that this results in substantial throughput improvements over the state of the art CTP in static settings, and superior delivery performance under dynamic settings such as external interference and mobile sinks.

For duty-cycled operation in long-lived sensor networks, we have tested BCP over asynchronous low power listening (LPL) MAC available in the current TinyOS 2.x distribution and verified that it is functional at moderate duty cycles (15-25%). However, for lower duty cycle operation, we find that the underlying LPL MAC will need to be modified to be more supportive of packet snooping. On the other hand, BCP should work quite well even at low duty cycles over synchronous sleep-based MAC protocols such as
S-MAC [147] which do not negatively affect snooping; we plan to explore these in the future.

One of the most exciting aspects of our work with BCP is the number of extensions available for future research and development, both by our group and others. We believe that BCP can be the basis of a comprehensive new high-performance cross-layer networking stack for wireless sensor networks. Some immediate extensions that we plan to pursue pertain to providing automated parameter adaptation (for protocol parameters such as $V$ and $\tau$). With a backpressure routing stack in place, it is very easy to implement transport layer congestion control on top that allows for the maximization of any concave source-rate utility function. We plan to do this in the future, similar to the backpressure-based transport layer optimizations implemented in [4, 121, 122]. We also plan to investigate if there are any further throughput gains to be obtained with MAC-layer prioritization based on the back-pressure weights (as has been explored in [4, 139]). Other desirable extensions include integrating BCP over a suitable multi-frequency MAC, exploring receiver diversity techniques, and network coding.

Finally, a direction of interest is to undertake a theoretical analysis of BCP over the TinyOS MAC for performance comparison with an optimized CSMA MAC [60, 102] to obtain bounds and guarantees on performance.
Chapter 4

Floating LIFO Delay Performance and Parameter Evaluation

4.1 Theoretical Analysis of Floating LIFO Queues

The floating LIFO queues employed in BCP address issues of scalability and packet delivery delay. In section 4.1.1, we will prove that using a data queue which grows \( O \left( \log^2(V) \right) \), we can bound the discard rate introduced by the floating queue as \( O \left( \frac{1}{V^{c_0 \log(V)}} \right) \) with constant \( c_0 \) that is \( O(1) \). Then in section 4.1.2 we provide a quantification of the delay advantages of LIFO queue service priority for a class of queues with special constant backlog properties, as occurs in BCP. We prove that as the arrival rate to these queues goes to zero, the average delay for serviced packets under FIFO discipline can be arbitrarily worse than the same delay metric under LIFO service priority.

4.1.1 Floating Queue Bound on Discard Rate

Recent work by Huang and Neely [50] takes an approach to the utility-delay challenge which does not require adoption of mixed Lyapunov functions. The authors, noting
that the steady state solution under performance optimal Lyapunov networking can be represented in a deterministic equivalent problem, prove that the backlog vector under quadratic Lyapunov function based algorithms deviates from the steady state with probability that is exponentially decreasing in the Euclidean distance from the steady state average. Using this knowledge, they propose the Fast Quadratic Lyapunov Algorithm (FQLA), a modification to the queueing procedure within nodes forming a QLA network. Two properties of FQLA make it challenging to implement in a system: it necessitates knowledge of, or a learning period to determine, the steady state optimal backlogs and it assumes infinite data queue availability. We will now use the Lagrange multiplier network gravity results from Huang and Neely [50] to provide discard bounds for finite data queues within our Floating Queue, requiring no knowledge of the steady state optimal queue backlogs.

4.1.1.1 Lagrange Multiplier Network Gravity Results of Huang and Neely

We will now give a brief description of the typical discrete time queueing framework, employed in the Quadratic Lyapunov Algorithm analysis of Huang and Neely [50], whose results are contained in this subsection.

\[ U_j(t + 1) = \max [U_j(t) - \mu_j(t), 0] + A_j(t) \] (4.1)

Where \( U_j(t) \) is the queue backlog in time slot \( t \), \( \mu_j(t) \) is the server capacity due to control decisions in time slot \( t \), and \( A_j(t) \) is the sum of exogenous and endogenous arrivals
to node \(j\) in time slot \(t\). Note that this definition assumes the generation of null packets when channel capacity exceeds queue occupancy.

**Definition 1.** Let \(B \in \mathbb{R}\) s.t. \(B \geq \| A(t) - \mu(t) \| \) \(\forall t\)

As noted in Chapter 3, optimal steady state queue backlogs \((U^*_V)\) for BCP scale with \(V\). This is a result of the link penalty (ETX) included in the performance optimal Lyapunov networking framework. Recall that the queue occupancy / penalty optimality tradeoff for this framework is \(O(V) - O(\frac{1}{V})\). That is, queues grow linearly with \(V\), while the penalty (per packet transmissions) approach the optimal like \(\frac{1}{V}\).

Huang and Neely [50] then transform the stochastic multi-hop network into an equivalent deterministic problem, and use Lagrange multipliers to form the dual problem. The authors bound \(P_r(D,m)\), the probability that in steady state any queue deviates from the steady state optimal backlog value by more than distance \(D + m\), for a given \(V\).

**Definition 2.** \(P(D,m)\)

\[
P(D,m) \equiv \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \Pr \{ \| U(\tau) - U^*_V \| > D + m \} \tag{4.2}
\]

**Definition 3.** \(P^{(r)}(D,m)\)

\[
P^{(r)}(D,m) \equiv \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \Pr \{ \exists j, |U_j(\tau) - U^*_{V,j}| > D + m \} \tag{4.3}
\]

Using the deterministic problem, and assuming that the stochastic problem has zero duality gap (conditions for closure are given in [50]), the authors find the following.
Theorem 1. (Theorem 2 of Huang and Neely [50]) If the dual function of the equivalent deterministic flow routing problem $q_0(U)$, as defined in [50], satisfies:

$$q_0(U^*_0) \geq q_0(U) + L\|U^*_0 - U\| \quad \forall U \succeq 0$$

for some constant $L > 0$ independent of $V$, then under QLA, for any $c > 0$:

$$P(D_1, cK_1 \log(V)) \leq \frac{c_1^4}{Vc}, \quad (4.4)$$

$$P^{(v)}(D_1, cK_1 \log(V)) \leq \frac{c_1^4}{Vc}, \quad (4.5)$$

where $D_1 = \frac{2B^2}{L} + \frac{L}{4}$, $K_1 = \frac{B^2 + BL/6}{L/2}$ and $c_1 = \frac{8(B^2 + BL/6)}{L^2} \frac{L/6}{L/2}$.

Huang and Neely [50] use this theorem to prove performance of their F-QLA algorithm. We will use their theorem to prove queue sizing and drop bounds for our Floating Queue.

4.1.1.2 The Floating Queue Algorithm

In essence, the Floating Queue is a data queue "floating" on top of a virtual data queue which shrinks or grows with data queue overflow or underflow. Figure 4.1 provides a startup sample path for a Floating Queue. Arrivals between time slots (i) and (ii) cause the data queue $D_i$ (shown as the dark segment) to grow to it’s maximum size. Subsequent arrivals between times (ii) and (iv) cause data queue overflow, enlarging the underlying virtual data queue $V_i$, which reaches maximum size at time (iv). Subsequent services
Figure 4.1: The sample path for a Floating Queue beginning operation. Cross-hatch bars
represent the data queue, gray bars represent the virtual data queue. Null packets are
generated when the data queue underflows.

completely deplete the data queue between time (iv) and (v), and further services between
(v) and (vi) result in reduction of the underlying virtual data queue, accomplished by
generating null packets. Note that null packets are simply data packets with a special
flag that informs the destination that the contents was lost in transit to the sink. Finally,
new arrivals between (vi) and (vii) resume the growth of queue $D_i$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U^V_i$</td>
<td>The steady state optimal backlog for a single queue at node $i$ under a Quadratic Lyapunov algorithm with parameter $V$.</td>
</tr>
<tr>
<td>$U_i(t)$</td>
<td>The backlog under QLA for a single queue at node $i$.</td>
</tr>
<tr>
<td>$\mu_i(t)$</td>
<td>Services under QLA for a single queue at node $i$.</td>
</tr>
<tr>
<td>$A_i(t)$</td>
<td>Endogenous and Exogenous arrivals under QLA for a single queue at node $i$.</td>
</tr>
<tr>
<td>$\mathcal{V}_i(t)$</td>
<td>The virtual data queue backlog in time-slot $t$.</td>
</tr>
<tr>
<td>$\mu^V_i(t)$</td>
<td>Services from virtual data queue $\mathcal{V}_i$ in time-slot $t$.</td>
</tr>
<tr>
<td>$A^V_i(t)$</td>
<td>Endogenous and Exogenous arrivals to virtual data queue $\mathcal{V}_i$ in time-slot $t$.</td>
</tr>
<tr>
<td>$D_i(t)$</td>
<td>The data queue backlog in time-slot $t$.</td>
</tr>
<tr>
<td>$\mu^D_i(t)$</td>
<td>Services from data queue $D_i$ in time-slot $t$.</td>
</tr>
<tr>
<td>$A^D_i(t)$</td>
<td>Endogenous and Exogenous arrivals to data queue $D_i$ in time-slot $t$.</td>
</tr>
</tbody>
</table>

Table 4.1: Definition of variables used in Floating Queue operation and proofs
Formally, we will preserve all arrivals and departures that would occur under a single queue QLA solution, by accounting for them in arrivals and departures to our Floating Queue sub-queues such that

\[ U_i(t) = D_i(t) + V_i(t) \quad \forall t. \]

In each time slot, weight calculations using advertised backlogs determine \( \mu_i(t) \) and \( A_i(t) \). We then allocate these services and arrivals to \( D_i \) and \( V_i \). Define \( R_i(t) \) to be the remaining data queue available within node \( i \) at the end of time-slot \( t \), so that

\[ R_i(t) = (D_{\text{max}} - D_i(t)) + \mu_i(t) \]

Then we can cleanly describe services of and arrivals to the data queue as follows:

\[
\mu_i^D(t) = \begin{cases} 
\mu_i(t) & \text{if } \mu_i(t) \leq D_i(t) \\
D_i(t) & \text{if } \mu_i(t) > D_i(t)
\end{cases}
\]

(4.6)

\[
A_i^D(t) = \begin{cases} 
A_i(t) & \text{if } A_i(t) \leq R_i(t) \\
R_i(t) & \text{if } A_i(t) > R_i(t)
\end{cases}
\]

(4.7)

After prioritizing data queue services and arrivals, we then give the remaining overflow (underflow) arrivals (services) to the virtual queue:
\[ \mu_i^Y(t) = \mu_i(t) - \mu_i^P(t) \]  \hspace{1cm} (4.8)

\[ A_i^Y(t) = A_i(t) - A_i^P(t) \] \hspace{1cm} (4.9)

To summarize, both services and arrivals are prioritized to the data sub-queue. After determining the service and arrival allocations to the sub-queues, discrete time sub-queues \( D_i(t) \) and \( V_i(t) \) are updated:

\[ D_i(t + 1) = \max \left[ D_i(t) - \mu_i^P(t), 0 \right] + A_i^P(t) \]  \hspace{1cm} (4.10)

\[ V_i(t + 1) = \max \left[ V_i(t) - \mu_i^Y(t), 0 \right] + A_i^Y(t) \] \hspace{1cm} (4.11)

We will now prove drop rate bounds for particular \( D_{\text{max}} \), a function of \( V \).

4.1.1.3 Preliminaries

In order to give a formal proof of the drop rate and data sub-queue sizing for fixed \( V \) under the Floating Queue algorithm, we will next derive some useful definitions and statements.

**Definition 4.** We will call queue i strongly stable if:

\[ \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[U_i(\tau)] < \infty \]  \hspace{1cm} (4.12)
Lemma 1. (Necessary Condition for Strong Stability) If a queue is strongly stable and either \( E\{A(t)\} \leq A_{\text{max}} \) for all \( t \), or \( E\{\mu(t) - A(t)\} \leq D_{\text{max}} \) for all \( t \), where \( A_{\text{max}}, D_{\text{max}} \) are finite non-negative constants, then:

\[
\lim_{t \to \infty} \frac{E[U(t)]}{t} = 0
\]

Definition 5. Let \( \bar{t}^B = (\bar{t}^B_0, \bar{t}^B_1, \bar{t}^B_2, \cdots) \) be the potentially finite, strictly increasing series of time-slots in which arrivals exceeded departures and the queue backlog rose from \( V_i(\bar{t}^B_j) < B \) to \( V_i(\bar{t}^B_j + 1) \geq B \).

Definition 6. Let \( \hat{t}^B = (\hat{t}^B_0, \hat{t}^B_1, \hat{t}^B_2, \cdots) \) be the potentially finite, strictly increasing series of time-slots in which departures exceeded arrivals and the queue backlog fell from \( V_i(\hat{t}^B_j) \geq B \) to \( V_i(\hat{t}^B_j + 1) < B \).

Definition 7. Let \( t^B \) be the strictly increasing and potentially finite series of time-slots resulting from the time-ordered sequence \( \bar{t}^B \cup \hat{t}^B \).

Proposition 1. Provided \( V_i(0) = 0 \), \( t^B \) will always begin with \( \bar{t}^B_0 \).

Proof. Assume that the first member of \( t^B \) is not \( \bar{t}^B_0 \). Then the first member of \( t^B \) is \( \hat{t}^B_0 \). But \( V_i(0) = 0 \), and \( \hat{t}^B_0 \) requires that queue backlog was reduced from \( V_i(\hat{t}^B_0) \geq B \) to \( V_i(\hat{t}^B_0 + 1) < B \). Therefore, at some time before \( \hat{t}^B_0 \) the queue backlog rose from 0 to greater than B. But this implies that there exists a \( \bar{t}^B_0 < \hat{t}^B_0 \). A contradiction. \( \square \)

Lemma 2. The series \( t^B \) will always consist of alternating members from \( \bar{t}^B \) and \( \hat{t}^B \).

Proof. Consider subsequent series members \( t^B_i, t^B_{i+1} \in t^B \). Without loss of generality, let \( t^B_i \in \bar{t}^B \). Assume that \( t^B_{i+1} \in \hat{t}^B \). This implies that no element \( \hat{t}^B_j \) satisfies \( t^B_i < \hat{t}^B_j < t^B_{i+1} \).
But this implies that the queue backlog dropped from $B$ to less than $B$ in time-slot $\bar{t}_i^B$ and in time-slot $\bar{t}_{i+1}^B$ without ever growing above backlog $B$ between time-slots $\bar{t}_i^B$ and $\bar{t}_{i+1}^B$. This is a contradiction, therefore $t_{i+1}^B \notin \bar{t}^B$.

By symmetry of argument, subsequent series members $t_i^B, t_{i+1}^B \in t^B$ cannot both be members of $\hat{t}^B$. Therefore, the series $t^B$ must consist of alternating members from $\bar{t}^B$ and $\hat{t}^B$. 

\[ \square \]

**Definition 8.** Define the queue backlog binary operator $\mathbb{1}_{a \leq V_i(t) \leq b}$:

\[
\mathbb{1}_{a \leq V_i(t) \leq b} = \begin{cases} 
1 & \text{if } a \leq V_i(t) \leq b \\
0 & \text{else}
\end{cases}
\]

**Theorem 2.** *(Bounded region of interest arrival-departure equivalency)* Given a discrete time queue $V_i(t)$ having binary arrivals and departures ($A_i^V(t) \in \{0, 1\}, \mu_i^V(t) \in \{0, 1\}$), initial zero backlog ($V_i(0) = 0$) and a bounded region of interest such that $a \leq V_i \leq b$, then it holds for any $t$ that:

\[
\sum_{\tau=0}^{t-1} A_i^V(\tau) \cdot \mathbb{1}_{a \leq V_i(\tau) \leq b} - \sum_{\tau=0}^{t-1} \mu_i^V(\tau) \cdot \mathbb{1}_{a \leq V_i(\tau) \leq b} \leq b - a + 1
\]

That is, total effective services of queue $V_i(t)$ within the region of interest lags total arrivals by no more than $b - a + 1$.
Proof. If \( \sum_{\tau=0}^{t-1} \mu_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} = \sum_{\tau=0}^{t-1} A_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} \). Then arrivals equal departures within the region of interest for time-slot \( \tau \), we can safely omit accounting for this time-slot.

If \( \sum_{\tau=0}^{t-1} \mu_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} > \sum_{\tau=0}^{t-1} A_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} \) then time-slot \( \tau \) is a member of \( \hat{t}V_i(\tau) \) because the queue was serviced from backlog \( V_i(\tau) \) to \( V_i(\tau) - 1 \).

Likewise, if \( \sum_{\tau=0}^{t-1} A_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} < \sum_{\tau=0}^{t-1} \mu_i^\tau(\tau) \cdot 1_{a \leq V_i(\tau) \leq b} \) then time-slot \( \tau \) is a member of \( \bar{t}V_i(\tau) \) because the queue received arrivals causing backlog increase from \( V_i(\tau) - 1 \) to \( V_i(\tau) \).

Therefore every time-slot \( \tau \) in which queue backlog changes within the region of interest is mapped to one of \( \hat{t}B \) or \( \bar{t}B \) for some \( B \).

Then for each backlog value \( a \leq B \leq b \) there exist a time series \( tB \) of arrivals and departures of the queue process to that backlog level, and which, per Lemma 2, has an imbalance of arrivals versus departures that is at most one. Within the region of interest \( a \leq B \leq b \), there can therefore be no more than \( b-a+1 \) more arrivals than departures. \( \square \)

Note that this proof can be extended to queues with non-binary arrivals and departures. The association of arrivals to departures becomes more notation-complex, as multiple backlog time series may be impacted by services / arrivals in a time-slot.

**Corollary 1.** (Bounded region of interest arrival-departure rate equality) Given a discrete time queue \( V_i(t) \) having binary arrivals and departures \( (A_i^\tau(t) \in \{0,1\}, \mu_i^\tau(t) \in \{0,1\}) \), initial zero backlog \( (V_i(0) = 0) \) and a bounded region of interest such that \( a \leq V_i \leq b \), it holds that:
\[
\lim_{t \to \inf} \sup_l \frac{1}{l} \sum_{\tau=0}^{t-1} A_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} = \lim_{t \to \inf} \sup_l \frac{1}{l} \sum_{\tau=0}^{t-1} \mu_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b}
\]

**Proof.** Using Theorem 2:

\[
\lim_{t \to \inf} \frac{1}{l} \left[ \sum_{\tau=0}^{t-1} A_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} - \sum_{\tau=0}^{t-1} \mu_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} \right] \leq \lim_{t \to \inf} \frac{1}{l} [b - a + 1]
\]

\[
\lim_{t \to \inf} \frac{1}{l} \left[ \sum_{\tau=0}^{t-1} A_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} - \sum_{\tau=0}^{t-1} \mu_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} \right] - \lim_{t \to \inf} \frac{1}{l} \left[ \sum_{\tau=0}^{t-1} \mu_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} \right] = 0
\]

And therefore

\[
\lim_{t \to \inf} \frac{1}{l} \left[ \sum_{\tau=0}^{t-1} A_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} \right] = \lim_{t \to \inf} \frac{1}{l} \left[ \sum_{\tau=0}^{t-1} \mu_l^V(\tau) \cdot 1_{a \leq V_l(\tau) \leq b} \right]
\]

\[\Box\]

**4.1.1.4 Bounding Floating Queue Discard Rates**

With preliminaries out of the way, we will now analyze the performance of our floating queues.

Note that discards of (potentially null) arriving data occur in queue update equation (4.11), where excess arrivals that overflow \(D_l(t)\) are allocated instead as virtual data to
\( \mathcal{V}_i(t) \). The discard rate of a floating queue is then equal to the admission rate to \( \mathcal{V}_i \). We therefore define the discard rate of node \( i \) (\( \eta_i \)) as:

\[
\eta_i \equiv \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E \left[ A^V_i(\tau) \right]
\]  

(4.13)

**Lemma 3.** If under QLA, the aggregate queue \( U_i(t) = D_i(t) + \mathcal{V}_i(t) \) is strongly stable, as defined in Equation 4.12, and either \( E\{A(t)\} \leq A_{\text{max}} \) for all \( t \), or \( E\{\mu(t) - A(t)\} \leq D_{\text{max}} \) for all \( t \), where \( A_{\text{max}}, D_{\text{max}} \) are finite non-negative constants, then:

\[
\limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E \left[ \mu^V_i(\tau) \right] = \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E \left[ A^V_i(\tau) \right] \equiv \eta_i
\]

That is, the time average expected discard rate is equal to the time average expected service rate of \( \mathcal{V}_i(t) \).
Proof.

\( U_i(t) \) is strongly stable \( \implies \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[U_i(\tau)] < \infty \)

\( \therefore \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[D_i(\tau) + V_i(\tau)] < \infty \)

but \( D_i(\tau) \geq 0 \)

\( \therefore \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[V_i(\tau)] < \infty \implies V_i \) is strongly stable

Now applying Lemma 1:

\( V_i \) is strongly stable \( \implies \lim_{t \to \infty} \frac{E[V_i(t)]}{t} = 0 \)

\( \therefore \limsup_{t \to \infty} \frac{1}{t} E \left[ \sum_{\tau=0}^{t-1} \mu_i^V(\tau) - \sum_{\tau=0}^{t-1} A_i^V(\tau) \right] = 0 \)

And by linearity of expectation:

\( \therefore \limsup_{t \to \infty} \frac{1}{t} E \left[ \sum_{\tau=0}^{t-1} \mu_i^V(\tau) \right] - \limsup_{t \to \infty} \frac{1}{t} E \left[ \sum_{\tau=0}^{t-1} A_i^V(\tau) \right] = 0 \)

\( \implies \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[\mu_i^V(\tau)] = \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E[A_i^V(\tau)] \)

\( \square \)

We will now bound the time average expected service rate of queue \( V_q \).

**Theorem 3.** (Discard rate of a floating queue)

Provided a floating queue having binary arrivals and departures with finite data storage
$O(\log^2(V))$, if the conditions of Theorem 1 hold and a steady state distribution exists for backlog process generated by QLA, then for large $V$ and $c_0$ which is $O(1)$:

$$P_{drop}^{FQ} = O\left(\frac{1}{V c_0 \log(V)}\right)$$

(4.14)

**Proof.** By Lemma 3, in order to bound the discard rate of our floating queue it is necessary and sufficient to bound the service rate of the virtual data queue ($V_i(t)$). We will do so by binning virtual queue services ($A_i^V(t)$) by virtual queue backlog $V_i(t)$ and bounding each region of operation, thereby bounding virtual queue services in all regions of operation.

Recall from the operation description of the floating queue, services of $\mu_i(t)$ will be pulled from $V_i(t)$ only if $D_i(t)$ contains insufficient backlog to satisfy the link rate. Because arrivals and services are assumed binary, at the time of servicing queue $V_i(t)$ it must be that $D_i(t) = 0$.

**Case 1: sub-queue $V_i$ is serviced while $V_i(t) < U_i^* - O(\log^2(V))$**

$$V_i(t) < U_i^* - O(\log^2(V))$$

But $D_i(t) = 0 \Rightarrow U_i(t) < U_i^* - O(\log^2(V))$

But from Theorem 1, the probability that $U_i(t) < U_i^* - O(\log^2(V))$ is bounded by $O\left(\frac{1}{V c_0 \log(V)}\right)$ for large $V$. Therefore, in this region of $V_i(t)$ the drop rate is also bounded by $O\left(\frac{1}{V c_0 \log(V)}\right)$ for large $V$. 

118
Case 2: sub-queue $V_i$ is serviced while $V_i(t) > U_i^* + O\left(\log^2(V)\right)$

$$V_i(t) > U_i^* + O\left(\log^2(V)\right)$$

But $D_i(t) = 0 \therefore U_i(t) > U_i^* + O\left(\log^2(V)\right)$

Again using Theorem 1, the probability that $U_i(t) > U_i^* + O\left(\log^2(V)\right)$ is bounded by $O\left(1/Vc_0\log(V)\right)$ for large V. Therefore, in this region of $V_i(t)$ the drop rate is again bounded by $O\left(1/Vc_0\log(V)\right)$ for large V.

Case 3: sub-queue $V_i$ is serviced while $U_i^* - O\left(\log^2(V)\right) \leq V_i(t) \leq U_i^* + O\left(\log^2(V)\right)$

In order to bound the expected time average rate of virtual queue services within this region of interest, we apply Corollary 1, which allows us to instead bound the rate arrivals to the virtual queue within this region of interest. Recall from the operation description of the floating queue, arrivals to $V_i(t)$ may only occur if the data queue $D_i(t)$ is full. Let the maximum occupancy of the data queue be $D^{MAX} = O\left(\log^2(V)\right)$

$$U_i^* - O\left(\log^2(V)\right) \leq V_i(t) \leq U_i^* + O\left(\log^2(V)\right) \cap D_i(t) = D^{MAX} \implies U_i(t) > U_i^* + O\left(\log^2(V)\right)$$

Therefore the arrival rate to queue $V_i(t)$ in the region $Q_i > U_i^* + O\left(\log^2(V)\right)$ is upper bounded by the arrival rate to $U_i(t)$ in the region $U_i > U_i^* + O\left(\log^2(V)\right)$, which is upper bounded by $O\left(1/Vc_0\log(V)\right)$ for large V.

Aggregate Drop Rate
Therefore we have bounded the drop rate for all possible time-slots:

\[ P_{\text{drop}}^{\text{FQ}} = \begin{cases} 
O \left( \frac{1}{V^c c_0 \log(V)} \right) & \text{if } V_i(t) < U_i^* - O \left( \log^2(V) \right), \\
O \left( \frac{1}{V^c c_0 \log(V)} \right) & \text{if } U_i^* - O \left( \log^2(V) \right) \leq V_i(t) \leq U_i^* + O \left( \log^2(V) \right), \\
O \left( \frac{1}{V^c c_0 \log(V)} \right) & \text{if } V_i(t) > U_i^* + O \left( \log^2(V) \right)
\end{cases} \]

Therefore the aggregate drop rate of the Floating Queue is bounded by \( O \left( \frac{1}{V^c c_0 \log(V)} \right) \) with data queue size \( O \left( \log^2(V) \right) \) for large \( V \).

\[ \square \]

4.1.2 Analysis of the LIFO Delay Advantage

In order to quantify the potential delay advantages of LIFO over FIFO within the performance optimal Lyapunov networking framework, we will consider some arbitrary queue \( Q_i(t) \) within the multi-hop BCP network. In order to conceptually simplify the analysis, we will omit the concept of Floating Queues.

**Definition 9.** Queue \( U_i(t) \) is defined as having a stabilized permanent backlog \( b_i^{\text{min}} > 0 \) if there exists a \( t^* \) such that for all \( t \geq t^* \), \( Q_i(t) \geq b_i^{\text{min}} \) and there exists an infinite sequence of time slots \( t^* \leq t_0 \leq t_1 \leq \cdots \) for which \( Q_i(t_j) = b_i^{\text{min}} \). Formally:

\[ \exists t^* \text{ s.t. } \forall t \geq t^* \quad Q_i(t) \geq b_i^{\text{min}} \cap \exists \left( t^* \leq t_0 \leq t_1 \leq \cdots \right) \text{ for which } Q_i(t_j) = b_i^{\text{min}} \]

(4.15)

**Definition 10.** The average delivered packet delay is defined as the average delay for packets passing through a queue but not trapped indefinitely within.
It is not useful to consider average delivered packet delay for arbitrary queueing systems, as without stability the volume of indefinitely trapped packets may grow to infinity. Within the context of a stabilized permanent backlog queue operating with LIFO service priority, however, this metric is meaningful. Though the average packet delay through such a queue is unchanged, we can improve the average serviced packet delay by permanently trapping and effectively discarding $b_i^{\text{min}}$ packets.

**Theorem 4. (The LIFO Delay Advantage for Constantly Backlogged Queues)** Let $Q_i(t)$ be a queue with stabilized permanent backlog $b_i^{\text{min}}$ and arrival rate $\lambda_i$. Then the time average delivered packet delay relationship under FIFO ($W_{FIFO}$) and LIFO ($W_{LIFO}$) queueing disciplines is exactly:

$$W_{FIFO}^i = W_{LIFO}^i + \frac{b_i^{\text{min}}}{\lambda_i} \quad (4.16)$$

**Proof.** $Q_i(t)$ has stabilized permanent backlog $b_i^{\text{min}} \Rightarrow (4.15)$ holds.

**Case LIFO:**

Under a LIFO discipline, any data arriving to find backlog greater than or equal to $b_i^{\text{min}}$ will be emptied infinitely often per (4.15). The oldest $b_i^{\text{min}}$ packets within the LIFO queue at time $t^*$ are trapped indefinitely, and therefore are not considered in calculation of average delivered packet delay. Beyond time $t^*$, the average delivered packet delay of the LIFO queue is therefore equivalent to the average packet delay of a LIFO queue operating with the oldest $b_i^{\text{min}}$ packets removed. Let $N_{i}^{\text{LIFO}}$ be the time average number of packets in LIFO queue $i$ after removal of the $b_i^{\text{min}}$ trapped packets.
Case FIFO:

Under a FIFO discipline, the average delivered packet delay is always equal to the average packet delay, as every arriving packet is eventually serviced. Let $N_{i}^{FIFO}$ be the time average number of packets in FIFO queue $i$.

As a result of the LIFO queue discipline, and the stabilized permanent backlog, we then find:

$$N_{i}^{FIFO} = N_{i}^{LIFO} + b_{i}^{min} \implies \frac{N_{i}^{FIFO}}{\lambda_{i}} = \frac{N_{i}^{LIFO}}{\lambda_{i}} + \frac{b_{i}^{min}}{\lambda_{i}}$$

$$\implies W_{i}^{FIFO} = W_{i}^{LIFO} + \frac{b_{i}^{min}}{\lambda_{i}}$$ (Little’s Applied Twice)

Where in the final step we use the fact that $N_{i}^{FIFO}$ is serviced with FIFO service priority and that the modified LIFO queue empties infinitely often, therefore Little’s Theorem applies for both queues.

In this chapter we first provide analysis of the drop rate and delay reduction capabilities of our floating LIFO queues. Next we perform parameter evaluation on the Tutornet testbed, validating the analysis through experimentation.

4.2 Floating LIFO Parameter Validation

In this section we validate our analysis against empirical results obtained from the same testbed and Backpressure Collection Protocol (BCP) code developed in [88]. It is important to note that these experiments are therefore not one-to-one comparable with the
analysis and simulations which we have previously presented. We note that BCP runs atop the default CSMA MAC for TinyOS which is not known to be throughput optimal, that the testbed may not precisely be defined by a finite state Markovian evolution, and finally that limited storage availability on real wireless sensor nodes mandates the introduction of virtual queues to maintain backpressure values in the presence of data queue overflows.

In order to avoid using very large data buffers, in [88] the forwarding queue of BCP has been implemented as a floating queue. The concept of a floating queue is shown in Figure 4.3, which operates with a finite data queue of size $D_{\text{max}}$ residing atop a virtual queue which preserves backpressure levels. Packets that arrive to a full data queue result in a data queue discard and the incrementing of the underlying virtual queue counter. Underflow events (in which a virtual backlog exists but the data queue is empty) results in null packet generation, which are filtered and then discarded by the destination.  

Despite these real-world differences, we are able to demonstrate clear order-equivalent delay gains due to LIFO usage in BCP in the following experimentation.

4.2.1 Testbed and General Setup

To demonstrate the empirical results, we deployed a collection scenario across 40 nodes within the Tutornet testbed (see Figure 4.2). This deployment consisted of Tmote Sky devices embedded in the 4th floor of Ronald Tutor Hall at the University of Southern California.

\footnote{The LIFO floating queue can be shown (through sample path arguments) to have a discard rate that is still proportional to $O\left(\frac{1}{\sqrt{c_0 \log V}}\right)$ with $c_0 = \Theta(1)$ derived in [50].}
In these experiments, one sink mote (ID 1 in Figure 4.2) was designated and the remaining 39 motes sourced traffic simultaneously, to be collected at the sink. The Tmote Sky devices were programmed to operate on 802.15.4 channel 26, selected for the low external interference in this spectrum on Tutornet. Further, the motes were programmed to transmit at -15 dBm to provide reasonable interconnectivity. These experimental settings are identical to those used in [88].

Figure 4.3: The floating LIFO queues drop from the data queue during overflow, placing the discards within an underlying virtual queue. Services that cause data queue underflows generate null packets, reducing the virtual queue size.
We vary $D_{max}$ over experimentation. In practice, BCP defaults to a $D_{max}$ setting of 12 packets, the maximum reasonable resource allocation for a packet forwarding queue in these highly constrained devices.

### 4.2.2 Experiment Parameters

Experiments consisted of Poisson traffic at 1.0 packets per second per source for a duration of 20 minutes. This source load is moderately high, as the boundary of the capacity region for BCP running on this subset of motes has previously been documented at 1.6 packets per second per source [88]. A total of 36 experiments were run using the standard BCP LIFO queue mechanism, for all combinations of $V \in \{1, 2, 3, 4, 6, 8, 10, 12\}$ and LIFO storage threshold $D_{max} \in \{2, 4, 8, 12\}$. In order to present a delay baseline for Backpressure we additionally modified the BCP source code and ran experiments with 32-packet FIFO queues (no floating queues) for $V \in \{1, 2, 3\}$. ²

### 4.2.3 Results

Testbed results in Figure 4.4 provide the system average packet delay from source to sink over $V$ and $D_{max}$, and includes 95% confidence intervals. Delay in our FIFO implementation scales linearly with $V$, as predicted by the analysis in [97, 91, 37]. This yields an average delay that grows very rapidly with $V$, already greater than 9 seconds per packet for $V = 3$. Meanwhile, the LIFO floating queue of BCP performs much differently. We have plotted a scaled $[\log(V)]^2$ target, and note that as $D_{max}$ increases the average packet delay remains bounded by $\Theta([\log(V)]^2)$.

²These relatively small $V$ values are due to the constraint that the motes have small data buffers. Using larger $V$ values will cause buffer overflow at the motes.
Figure 4.4: System average source to sink packet delay for BCP FIFO versus BCP LIFO implementation over various V parameter settings.

Figure 4.5: System packet loss rate of BCP LIFO implementation over various V parameter settings.
These delay gains are only possible as a result of discards made by the LIFO floating queue mechanism that occur when the queue size fluctuates beyond the capability of the finite data queue to smooth. Figure 4.5 gives the system packet loss rate of BCP’s LIFO floating queue mechanism over $V$. Note that the poly-logarithmic delay performance of Figure 4.4 is achieved even for data queue size 12, which itself drops at most 5% of traffic at $V = 12$. We cannot state conclusively from these results that the drop rate scales like $O\left(\frac{1}{\log V}\right)$. We hypothesize that a larger $V$ value would be required in order to observe the predicted drop rate scaling. Bringing these results back to real-world implications, note that BCP (which minimizes a penalty function of packet retransmissions) performs very poorly with $V = 0$, and was found to have minimal penalty improvement for $V$ greater than 2. At this low $V$ value, BCP’s 12-packet forwarding queue demonstrates zero packet drops in the results presented here. These experiments, combined with those of [88] suggest strongly that the drop rate scaling may be inconsequential in many real world applications.

In order to explore the queue backlog characteristics and compare with our analysis, Figure 4.6 presents a histogram of queue backlog frequency for rear-network-node 38 over various $V$ settings. This node was observed to have the worst queue size fluctuations among all thirty-nine sources. For $V = 2$, the queue backlog is very sharply distributed and fluctuates outside the range $[11 - 15]$ only 5.92% of the experiment. As $V$ is increased, the queue attraction is evident. For $V = 8$ we find that the queue deviates outside the range $[41 - 54]$ only 5.41% of the experiment. The queue deviation is clearly scaling sub-linearly, as a four-fold increase in $V$ required only a 2.8 fold increase in $D_{max}$ for comparable drop performance.
Figure 4.6: Histogram of queue backlog frequency for rear-network-node 38 over various V settings.
Chapter 5

Rate Control and Dynamic Routing

As a result of stochastic throughput-optimal source rate utility optimization by Neely et al. in [97] and virtual queue mechanisms to maintain time average utility or penalty functions in [92], an entire optimization framework has been derived [97, 91]. Note that an alternative framework exists, conceived by Stolyar [126], but which we omit in order to avoid an abundance of alternative notation and the risk of confusion. We will discuss now our work to explore the rate control potential of this framework, specifically in the context of translation to system deployments.

In order to accomplish source rate control, the framework of [97, 91] introduces the concept of an infinite packet reservoir, from which packet admission decisions are made. These admission decisions generally require only knowledge of the source’s local forwarding queue backlog, and are therefore very layer-friendly in implementation. Further, they tend to be quite simple, adding little or no complexity to the system implementation. This positive feature happens to come for free, as the congestion information is already being generated by the lower layers (scheduling, routing, resource allocation of backpressure stacks below the protocol layer).
Finally, we would like to highlight that the predominance of WSN rate control protocols have been evaluated on quasi-static or static routing frameworks. We have demonstrated very strong performance of backpressure routing in systems with dynamics (see Chapter 3), and will be evaluating source rate control here on the same dynamic routing framework. The degree of per-packet routing flexibility and associated source rate control performance is therefore a novel contribution of our work.

In this chapter we will first discuss in Section 5.1 the system merits of max-min and proportional rate control objectives, with respect to Wireless Sensor Networks. We’ll then give a review of the theoretical control decisions dictated by each utility function, per the work of Neely et al. [97, 91], in Section 5.2. In Section 5.3 we’ll then describe the source rate control module we wrote for TinyOS, which interfaces seamlessly with the BCP routing stack written in Section 3. We’ll give experimental performance results of the stack in Section 5.6, then discuss the implications of our findings in Section 5.10

5.1 Selecting a Source Rate Utility Function

Investigation of linear utility maximization is rather fruitless, as discussed in Section 2.5.1. The optimal control decisions for these utilities in multi-hop topologies draw maximal rate from only a few select sources, omitting all others. This does not yield fair or interesting rate allocations. We will therefore investigate proportional fair and max-min fair source utility functions. Recall that proportional fair rate controllers aim to maximize:

\[ \max \sum_i \ln (r_i) \]  

(5.1)
Due to inter-flow interference and the concavity of the natural log, proportional fair solutions typically collect from all nodes in the network to at least a minimal degree. This is due to the diminishing returns for high one-hop data rates. As such, proportional fair utility maximization provides at least some degree of fairness, where the linear utility maximization objectives do not.

With the objective of fairness above all else, max-min fair allocations have become the baseline against which fairness metrics are generated [57, 110]. It is often difficult to analytically evaluate max-min objectives, we therefore will use a alpha-fair utility function to approximate max-min fair:

\[
max \sum_i (r_i)^{\frac{1}{\alpha}}
\]  

Note that as \( n \to \infty \) the alpha-fair solution goes to the max-min fair allocation. Practically, we will not be able to operate our backpressure stack with \( n \) arbitrarily large. We will demonstrate empirically, however, that even for small alpha (e.g. \( \alpha=4 \)) the alpha-fair solution strongly motivates fairness when compared to the performance of the proportional fair objective.

### 5.2 The Theory Behind Backpressure Source Rate Control

One strength of the framework resulting from the stochastic network optimization approach described in [97, 91] is the clean separation of control decisions between the protocol and network layers. Therefore, we need only describe here the source admission rate
controller design, as dictated by the analytical framework. The lower layers will consist of an un-modified BCP stack, described in Chapter 3.

In the theoretical literature, time is assumed to be slotted. The admission controller is assumed to have access to an infinite pool of backlogged application data, and determines how much volume \( R_i(t) \) should be admitted by node \( i \) in time slot \( t \). This admitted traffic is placed directly within the local forwarding queue, where it is operated on by the lower layers of the backpressure framework discussed previously (routing, scheduling, and resource allocation). Previous publications have explored the implementation of proportional fair source rate controllers in the backpressure framework [97, 91]. The optimal admission controller sets \( R_i(t) \) like:

\[
R_i(t) = \frac{V_{prop}}{U_i(t)} \tag{5.3}
\]

Where \( U_i(t) \) is node \( i \)'s backlog in time slot \( t \). Similarly, the prior work in alpha-fair source rate control has established the following \( R_i(t) \) control decision [121]:

\[
R_i(t) = V_{alpha} \cdot \frac{1}{[U_i(t)]^{\alpha}} \tag{5.4}
\]

Simply observing the rate control decisions specified in 5.3 and 5.4, we can draw some practical system implications. Though from a theoretical standpoint the \( V_{prop}/V_{alpha} \) parameter can be made very large, that is unreasonable for realistic systems. An implication is that for even lightly loaded networks (i.e. sources having a queue backlog of only one packet) both utility functions will achieve an admission rate of at most \( V_{prop}/V_{alpha} \).
Therefore, selection of this tuning parameter immediately inhibits the maximum system throughput.

A second observation is with regard to the scalability or breadth of system diversity for which a specific parameter setting provides good performance. If we are optimizing for proportional-fair allocations and $V_{prop} = 10$ provides both reasonable utility and queue backlogs, then a system that achieves only half the per-source rate will stabilize only once the queue sizes are twice as large. While this may seem adverse, we note that the alpha-fair objective is even more sensitive to system scaling. With $\alpha = 4$, halving the per-source rate requires sixteen times larger queues! Clearly this will provide scaling problems in a real system, as we’ll demonstrate empirically in Section 5.6.

Finally, it is important to note that by adding proportional or alpha-fair source rate controllers we have introduced a multi-objective optimization problem. There will be tension between our ETX-minimizing objectives (prioritized by $V_{ETX}$) and our source rate optimization objective (prioritized by $V_{prop}$ or $V_{alpha}$). We will show in Section 5.6 that as $V_{ETX}$ is increased the utility achieved decreases for fixed $V_{prop}/V_{alpha}$. Likewise, as $V_{prop}/V_{alpha}$ are increased the system transmissions per packet increase for fixed $V_{ETX}$. There is therefore an entirely new optimization problem, which we do not address here. How does one best tune these opposing parameters?

### 5.3 Implementation Details in BCP

We have implemented the source rate controller for BCP as a module that resides atop the BCP implementation of Chapter 3. With the dual goal of simplifying the rate control
mechanism (avoiding token bucket operation) and de-correlating potentially deterministic and synchronized packet generations, our rate controller design buffers send requests and generates admissions to BCP that have exponential inter-arrival times.

The rate controller implementation in TinyOS 2.x contains a pool for storage of at most six packets, to which application layer admission requests are stored and from which admissions are made into the BCP stack below. When the application layer attempts a send request to the BCP rate controller, the message pool is checked to see if room exists. If the source has been generating traffic above the rate determined utility optimal by the BCP stack, this pool will fill up. In this event, the application layer receives an EBUSY error response to admission requests.

The BCP rate controller determines the duration that the head-of-line packet must wait before admission into the BCP framework. This is accomplished by first computing the time average rate dictated by the backpressure framework (Equation 5.3 or 5.4), then generating an exponential wait time on this distribution. When the timer fires, the head of line packet is admitted into the BCP routing stack and if another packet exists a subsequent inter-arrival time is calculated. When compared to alternative rate control mechanisms such as WRCP [120] and IFRC [113], the backpressure framework rate controller is extremely simple. The primary complexity actually results from the computational capability of these motes to generate exponential random variables and calculate the time average functions.

To generate an exponential random variable with time average rate $\lambda$, we use rejection sampling (also known as the accept-reject algorithm) [114]. This algorithm’s primary drawback is the random (potentially high) number of uniform random variable samples
required before returning an exponential sample. We handle this by an abort mechanism, which causes minor tail truncation in our exponential distribution.

While the average rate computation is simple for our proportional fair rate controller (recall it is simply the ratio of \( V \) to the local queue backlog, in Equation 5.3). Things are not as simple for max-min fair, however. Recall that we approximate the max-min fair utility function by equation 5.2. Clearly we cannot afford to take \( \alpha \to \infty \). In the work of Sridharan et al. [113] the authors used \( \alpha = 8 \) to approximate max-min fair allocations. Subsequent alpha-fair rate controllers of [119] are implemented with \( \alpha \in \{5, 7\} \). These prior deployments did not rely on backpressure stacks for dynamic routing, nor did they operate under the highly constrained forwarding queue size of BCP (12 packets). We therefore find a need to ease the \( \alpha \) coefficient in order to reduce the queue sizes required in staunching source admissions, and to allow BCP’s ETX-minimization efforts to remain effective.

We therefore chose to approximate the max-min fair utility function by using \( n = 4 \), requiring we take the fourth root of the queue size. Unfortunately, such an operation is not natively supported in these processors. We therefore apply Newton’s Method to approximate the fourth root.

### 5.4 Empirical Max-Min Fair Rate

We ran experiments on 20 and 40 motes over the BCP stack while omitting rate control and using \( V_{ETX} = 2 \). Traffic was generated with exponential inter-arrival times (poisson). To determine the max-min fair rate allocation, we repeatedly increased the poisson traffic
rate $\lambda$ until the network delivery ratio dropped substantially and per-packet transmissions begin to exhibit high variance. Figure 5.1 provides the both an acceptable source rate (3.75 pps / source) and a marginally higher source rate which causes failure (4.1 pps / source). Note that for the higher data rate, many rear network sources are achieving delivery ratios only in the low 90% range, a five to ten percent drop. Furthermore, the 95 percent confidence intervals on per-packet transmissions begin to widen, indicating that the network congestion has begun to substantially impact link reliability.

We would like to discuss the source of the network losses for $\lambda = 4.1$. Throughout the entire 25 minute experiment, the sink received zero null packets. This means that no queue in the network fluctuated beyond the 11 packet forwarding buffer size, indicating that the floating queue mechanism of Chapter 3 is proving highly effective even as the system begins to fail. Additionally, this indicates that there are no persistent packet losses in the system due to queue tail drops. Observations of node logs indicate that at times a transmitting node perceived successful forwarding of a packet (indicating an ack was received) while the receiving node does not register the arrival of the packet. This has been referred to as a "false ack" by prior literature, though the incidence documented was much lower. In [135] the authors carefully documented a link-layer failure rate of 0.16% per transmission. In our experiment, rear nodes are experiencing on average four transmissions per packet, which would indicate a link-layer loss rate of less than 1%. False acks have also been documented at less than 0.1% in [124]. We hypothesize that our promiscuous radio settings, high data rates and poor link selection during network collapse are causing false acks at a rate ten times the previous empirical results, perhaps as the bus between the radio and processor becomes overloaded by retransmission attempts.
Figure 5.1: Per-source goodput, average packet transmission count, and delivery percentage for $V_{ETX} = 2$ and $\lambda \in \{3.75, 4.1\}$ PPS/Source, poisson traffic. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals. Note that beyond 3.75 packets per second per source, system stability is compromised.
It is worth mentioning that determining this max-min fair rate proved more difficult than expected. We found it often the case that a particular source rate (say $\lambda = 4.0$) would prove stable for many minutes before a burst of arrivals pushed the wireless links toward congestion. What then happened was that a rear-network node would experience substantial queue growth, which then overwhelmed the ETX penalties on lesser quality outbound links for that node. The node would therefore select some longer distance link due to the excessive queue backlog, further congesting the wireless domain and reinforcing the network failure. This leads to system-wide lower delivery ratios and effective congestive collapse. This should not occur in theory, but because we have no throughput optimal and centralized scheduler making control decisions we rely on the $V_{ETX}$ penalty mechanism to aid in good link selection. Destabilized queues overwhelm this safety technique and cause BCP to fail. This effect highlighted for us the usefulness of source rate controllers, which we will discuss next.

5.5 Rate Control Experimental Setup

All sources generate sample data at a rate of 10 Hz, deterministically. This somewhat arbitrary rate assured that the application layer would have a reasonable data supply for the rate controller to operate on. Unless otherwise noted, we use motes 1-20 (or 1-40) in the Tutornet Testbed, with a power level of 5 (-18 dBm). Channel 26 was leveraged in order to provide a low external interference environment with high reproducibility. We vary across experiments both $V_{ETX}$ and $V_{alpha}$ or $V_{prop}$, the constant parameters which trade queue size for transmission minimization and utility maximization respectively.
Experiments were run for 25 minutes each. The total packet size, including CC2420 header, BCP header and the payload field, comes to 34 Bytes.

5.6 Alpha-Fair Approximates Max-Min Capacity

First we will discuss the degree to which alpha-fair approximates Max-Min for $\alpha = 4$ in 20 mote experiments. Figure 5.2 provides per-source performance statistics for both our empirically derived Max-Min rate of 3.75 pps and for our rate controller with $V_{\alpha} = 5.4$. We note that at the rear of the network, queue sizes reduce our source rates slightly below the Max-Min empirical rate. In all cases, however, the under-performance is within 5% of the Max-Min source rate. Due to the difficulty in precisely detecting and achieving system overload it is difficult to make rigorous comparison. It may be possible to extract another 5% goodput from the Max-Min empirical rates. Regardless, the performance of the alpha-fair controller is clearly very close to the maximum sustainable Max-Min fair rate over BCP.

Moving on to 40 mote experiments, we begin to observe the challenges posed by small $\alpha$ settings, specifically in this multi-objective optimization. Figure 5.3 provides per-source performance statistics for both our empirically derived Max-Min rate of 1.33 pps and for our rate controller with $V_{\alpha} = 2.15$. The performance gap between empirically derived max-min fair rate and the BCP rate alpha-fair rate controller grows to 20% for rear network nodes in this experiment. The root cause is two-fold, but solutions are a balancing act which may be topology-dependent. Note that steady-state queue backlogs for rear network nodes are a function of both ETX penalties along the path(s) to the sink.
Figure 5.2: Per-source goodput, average packet transmission count, and delivery percentage for both alpha-fair source rate utility with $V_{ETX} = 2$ and $V_{alpha} = 5.4$ and the empirically derived max-min fair experiment. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.
(scaled by $V_{ETX}$) and the system loading by the source rate controller (scaled by $V_{alpha}$).

Therefore, rear-network nodes find themselves with substantially higher backlogs than one or two hop nodes from the sink. With such a small alpha (recall, $\alpha = 4$), there can be substantial throughput scaling as a result.

![Figure 5.3: Per-source goodput, average packet transmission count, and delivery percentage for both alpha-fair source rate utility with $V_{ETX} = 2$ and $V_{alpha} = 2.15$ and the empirically derived max-min fair experiment. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.](image)

To demonstrate this fact, we note that for rear network nodes in the 20 mote experiment the average transmissions per packet was only four. Rear network nodes therefore had a penalty backlog of at least 8 packets (due to $V_{ETX} = 2$), upon which congestion would add backlog. When compared with one hop nodes with a penalty backlog of 2
packets, we note that $2^{\frac{1}{4}} = 1.19$ while $8^{\frac{1}{4}} = 1.68$. Therefore, if one considers only the ETX penalty, we would expect rear-network nodes to achieve approximately 71% of the one-hop node admission rate. This matches well with empirical results of Figure 5.2 in which rear nodes achieved 76% of the one-hop admission rate. In the 40 mote experiment of Figure 5.3, rear nodes were as much as six transmissions from the sink, resulting in ETX penalties of 12 packets. Then we have $12^{\frac{1}{4}} = 1.86$, or an admission rate ratio versus one-hop nodes of 64%. In actuality, we observe admission rates to be 55% for the rear network nodes, compared to one-hop nodes. This gap can be partially attributed to rear network delivery losses of 5%.

This admission gap is not easily solved, as there is parameter tension between $V_{ETX}$ and $\alpha$. If one increases $\alpha$ to "flatten" the queue size penalty, fairness between one-hop and rear network nodes would increase but queue sizes would grow substantially before the higher order root function staunched flow. Furthermore, these larger queue sizes may negate built-in ETX penalties controlled for by $V_{ETX}$, causing poor link selection. If one then reactively increased $V_{ETX}$ we would find queue sizes in the rear of the network would grow disproportionately to one-hop queues, again enhancing the admission disparity for a given $\alpha$. Ultimately, a reasonably small $\alpha$ is needed in order to obtain reasonable real-world performance, and this will impact the fairness of the source rates under BCP’s ETX penalty as seen in Figure 5.3.
5.7 Tension Between $V_{ETX}$ and $V_{alpha}$ Parameters

We will now explore the impacts of $V_{ETX}$ and $V_{alpha}$ selection, specifically the interplay between the two optimization objectives. Figure 5.4 provides the per-source performance statistics for thirty-nine sources with $V_{alpha} = 2.5$ and $V_{ETX} \in \{1, 2\}$. This is an aggressive rate controller setting for $V_{alpha}$, as evident by the 86% delivery ratios for some sources in the case where $V_{ETX} = 1$. The impact of enlarging $V_{ETX}$ from 1 to 2 is very substantial with respect to average transmissions per packet and delivery ratio. Interestingly, any difference in rate achieved per source is not statistically significant. We attribute this to the 5-10% increase in delivery ratio for rear nodes under higher $V_{ETX}$, which counteracts the reduced source rate decisions.

If we move to a less aggressive admission controller parameter, say $V_{alpha} = 1.88$, we begin to observe the impacts of $V_{ETX}$ on fairness. Figure 5.5 gives per-source performance statistics for 40 mote experiments with $V_{ETX} \in \{1, 2\}$ and $V_{alpha} = 1.88$. Under the lesser $V_{ETX}$ setting of 1, the ratio of minimum source rate to maximum source rate is 66%, while for $V_{ETX}$ equal to 2 the gap widens to 55%. The larger queue penalties at the rear of the network are adversely affecting their admission rates.

5.7.1 Proportional Fair Experimental Results

To briefly demonstrate and visualize the comparative fairness of a proportional fair rate controller, we ran both 20 and 40 mote experiments with $V_{ETX} = 2$. The per-source performance statistics, compared to the alpha-fair solutions, are provided in Figures 5.6 and 5.7. Empirically we have found that as $V_{prop}$ is raised beyond the point at which
Figure 5.4: Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with $V_{ETX} \in \{1, 2\}$ and $V_{alpha} = 2.5$. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.
Figure 5.5: Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with $V_{ETX} \in \{1, 2\}$ and $V_{alpha} = 1.88$. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.
one-hop nodes achieve the max-min empirical rate, interference prevents further increase. Furthermore, due to the relatively unfair allocations of proportional fair, when compared with max-min, the rear network nodes are then achieving substantially lower goodput than the alpha-fair solutions.

Figure 5.6: Per-source goodput, average packet transmission count, and delivery percentage for Proportional-fair source rate utility with $V_{ETX} = 2$ and $V_{prop} = 20$, $V_{alpha} = 5$. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.8 Rate Controller Parameter Sensitivity

In previous static-routing exploration of backpressure source rate control by Sridharan et al. [121], the authors noted the sensitivity of the backpressure utility performance
Figure 5.7: Per-source goodput, average packet transmission count, and delivery percentage for Proportional-fair source rate utility with $V_{ETX} = 2$ and $V_{prop} = 5$, $V_{alpha} = 2.15$. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.
to the selection of the $V_{util}$ parameter and topology or traffic patterns. We therefore explored this sensitivity again, but now with a dynamic backpressure routing stack (BCP) and the inherent multi-objective optimization. Figure 5.8 provides per-source goodput performance for a 20 mote experiment with $V_{ETX} = 2$, $V_{alpha} \in \{2.5, 5\}$. Note that for the 40 mote experiments provided earlier, $V_{ETX} = 2$ and $V_{alpha} = 2.15$ provided optimal max-min source performance. Clearly this smaller topology performs sub-par with the settings that were tuned for a 40 mote all-source deployment.

Figure 5.8: Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with $V_{ETX} = 2$ and $V_{alpha} \in \{2.5, 5\}$. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.
Parameter adaptation will be an important component to a realistic future rate controller. We note that parameter selection is an important feature of much prior rate control literature, and many de-centralized protocols require system knowledge for proper parameter settings (e.g. WRCP [120] requires system average retransmissions). Centralized or end-to-end rate control techniques (e.g. RCRT [104]) are better able to learn about the network as a system, making parameter tuning a simpler manner (if needed at all).

5.9 Comparison with WRCP

Aside from a comparison of the rate controller performance versus our own empirically derived max-min fair rate estimates, we also run our alpha-fair rate controller over BCP for a node set, power setting, and packet size used in evaluation of IFRC and WRCP [119]. Figure 5.12 of [119] and our own Figure 5.9 were both generated under identical node, power, and packet size settings. A very substantial source of the greater alpha-fair rate in our experiment was the freedom of BCP to optimize the routing selection, whereas in [119] the routing was pinned and fairly conservative (resulting in a high hop count to the sink). Still, our backpressure rate controller atop BCP achieves a max-min fair rate that is just over twice that achieved by either IFRC or WRCP in [119] (and for some nodes is more than four times the rate of [119]).
Figure 5.9: Per-source goodput, average packet transmission count, and delivery percentage for alpha-fair source rate utility with $\alpha = 4$, $V_{ETX} = 2$ and $V_{alpha} = 2.5$. Unlike prior sections, we enlarge our packet to 40 Bytes for proper comparison with the IFRC/WRCP results of [119]. Additionally, the sink node is 29. As the per-packet transmission count is an average over per-packet arrival statistics, we provide 95% confidence intervals.

5.10 Summary and Discussion

We feel it is important to highlight the extreme dynamics of the multi-path routing protocol on which our BCP rate controller is operating. The dynamic capabilities of BCP were demonstrated in Chapter 3, the capability of the rate controller to run atop such variable routing is a potential strength which deserves further exploration in the future. Figure 5.10 provides the average number of packets forwarded between routing modifications by each node in the 40 mote high data rate alpha-fair experiment. Much of the rear of the network forwards fewer than ten packets between routing changes, while a handful averages only five successful packet transmissions before re-routing to a new neighbor. Yet the rate controller results of Figure 5.3 provide strong max-min fair approximate results even over this variability (or perhaps thanks to this variability).
We note that existing rate control protocols discussed in Section 2.5 were not tested over, and in many cases did not support, dynamic routing or multi-path routing. We noted in Chapter 3 that by supporting multi-path routing and reducing queue drops, BCP enhanced the max-min fair rate achievable on Tutornet. Support for these features, and the enhanced capacity region that results, are naturally built into the backpressure rate control framework we leveraged in this work.

Further, the simplicity of the backpressure rate control framework (from a systems standpoint) is a substantial asset in low power WSN. In total, BCP with the rate control engine requires only four system design parameters (Max Forwarding Queue Size, $V_{ETX}$, $V_{prop}$, and the EWMA link estimator weight). Compare this with a typical rate control stack, say IFRC [113], which has 8 such tunable parameters. The difficulty in validating
large sets of system parameters is substantial. In part because the backpressure framework supports a complete stack solution, the per-packet overhead required to implement features up to and including rate control is minimal. Both WRCP [120] and IFRC [113] leverage packet snooping mechanisms, like BCP, to disseminate control information. The packet header overhead of WRCP (14 Bytes) and IFRC (26 Bytes) are substantially larger than that of BCP (8 Bytes).

Finally, a major contribution of this backpressure rate control exploration is to explore the $V$ scaling requirements of some backpressure utility functions, particularly in a multi-objective optimization framework. Interestingly, not all utility / penalty functions face the same scaling challenges. The transmission penalty mechanism of BCP, for example, has been found to scale well across source rate and topology size (See empirical results of Chapter 3). Yet both proportional fair and alpha-fair source rate optimization have been shown to scale poorly in static routing applications [121] and now in dynamic routing frameworks. These findings strongly support the need for $V$ parameter adaptation in source rate optimization for real system deployments. The challenge lies in incorporating $V$ adaptation without hamstringing the strong backpressure stack performance in systems with strong dynamics, a strength of backpressure routing we highlighted experimentally in Chapter 3.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In implementing the Backpressure Collection Protocol (BCP), a dynamic backpressure routing collection protocol for WSN, we discovered a number of theory-to-system barriers. We described in Chapter 3 our work in creating BCP, key difficulties were packet looping and subsequent losses due to link layer failures, backpressure failure due to finite queue overflows, and severe packet delivery latencies at low source rates. We addressed each of these challenges to a degree sufficient to prove backpressure routing competitive with existing tree routing techniques, even in static settings. To remove packet looping we used expected transmission count (ETX) link penalties within the stochastic network optimization framework. Addressing finite queue overflows, we leveraged recent theoretical queue gravity results and implemented floating data queues. And through novel application of LIFO queueing priority we demonstrated a reduction greater than 98% in packet delivery latency for low source rates.
Our work in Chapter 3 motivated our subsequent theoretical analysis of the LIFO delay advantage, which we explore in Chapter 4. We prove that for data queue sizing which is $O(\log^2(V))$, our floating queue achieves a packet discard rate that scales like $O\left(\frac{1}{V^{\log(V)}}\right)$ for networks satisfying the assumptions of the queue gravity work presented in Huang and Neely [50]. We note that recent work by Huang et al. has substantially strengthened this theoretical result for LIFO queues in backpressure frameworks [49]. We then provide empirical testbed validation of this scaling property using our BCP deployment.

Our third contribution is detailed in Chapter 5, where we move up one layer to implement source rate utility maximization atop our BCP implementation. We explore for the first time the performance of backpressure rate control atop a backpressure dynamic routing stack (here BCP), and the real system impacts of multi-objective optimization within the stochastic network optimization framework. The optimization goals provided here are in tension; we aim to both minimize packet transmissions (ETX) and maximize aggregate source utility. We show empirically that for suitable selection of the optimization parameters we can approximate max-min fair source rate allocations within 95% of maximum for 20 nodes and 80% of maximum for 40 nodes. Through empirical evaluations of proportional fair objectives we also demonstrate the flexibility of the backpressure stack to accept alternative source rate utility functions.
6.2 Future Work

In this work, our main focus has been the design of a dynamic backpressure routing protocol and rate controller for wireless sensor networks. Our investigation has sparked many ideas that can form the basis for future research, which we will now discuss in some detail.

6.2.1 Performance Under Node Mobility

Node mobility is a very broad term with implications that vary according to the extent and type of mobility. In Chapter 3 we demonstrated sink mobility, in which the destination to which traffic was directed would rapidly move about the testbed. As we saw in Table 3.1 of Chapter 3, delivery ratio and average transmission count for BCP improved when compared with the static network tests. The same cannot be said for the tree-based collection algorithm (CTP). We believe that stateful tree-regeneration approaches to sink mobility have higher control overhead and cannot support the degree of mobility possible under backpressure algorithms which have a much finer granularity of forwarding decisions. These results hint that Backpressure routing protocols such as BCP may be capable of demonstrating enhanced system capacity in highly mobile networks.

In exploring the failure the tree-based protocols under extreme sink mobility, we were struck by a heretofore under-emphasized advantage of backpressure algorithms as described by Tassiulas and Ephremides [128]. Under extreme network dynamics, these original backpressure algorithms converge to routing and forwarding decisions while actively forwarding packets and carefully balancing per node queue backlogs. Essentially,
data is forwarded simultaneously with incremental routing knowledge updates. This con- trasts sharply with global shortest path algorithms for which the traditional time-scale of convergence is violated: resulting in wild queue size variation, packet drops, and extraneous transmissions as data is forwarded aggressively over unconverged perceived shortest paths.

This insight raises some level of concern over recent extensions to backpressure that leverage explicit shortest-pat computations for delay reduction. For instance, Ying et al. describe a max-hop-count guarantee backpressure algorithm in [149] which leverages minimum hop count knowledge to reduce queue storage and speed up learning. These algorithms reduce delay by leveraging information derived from an external shortest-path algorithm, and we hypothesize that such attempts to reduce delay will come at a cost of reduced responsiveness to mobility. We believe there exists an important open question here. Can it be proven that there is a fundamental tradeoff between data latency and the loss rate of data during network evolution, under finite data queue assumptions? Is the most stable routing algorithm one that transfers data only at the speed of routing updates and knowledge? This is a question somewhat tangential to the mobile node scalability results of Grossglauser and Tse [40].

6.2.2 Handling Traffic Dynamics

Theoretical literature has just begun to address finite duration dynamic flow settings, for which the traditional max-weight-matching schedule is non-optimal [133]. The analysis and solutions for this setting proposed in [133, 82], however, are limited to single-hop
networks and assume that there are an infinite number of finite flows and that queues are unbounded.

In prior work we have implemented and evaluated rate utility optimization over a static tree-routing mechanism in wireless sensor networks [121]. We found empirically that while backpressure based rate utility optimization works very well for static long-duration flows, it can fail in the presence of dynamics involving short-duration flows. The key is that in a real network, choosing a V-parameter that is large enough to ensure throughput optimality for a wide range of traffic settings results in queue overflows. On the other hand, maintaining a more conservative setting of the V parameter to avoid queue drops will place an artificial cap on maximum utility.

These complementary findings in theoretical and empirical settings suggest that the problem of efficiently handling traffic dynamics in the form of finite short-duration flows is still wide open. Future investigation into this question could develop an appropriate solution (e.g. live V parameter adaptation), as this is of great importance for many practical wireless network applications and scenarios.

6.2.3 Receiver Diversity

The demonstration of backpressure receiver diversity techniques in a real testbed environment, subjected to dynamics in node mobility and link quality, remains unexplored. We believe there are non-trivial challenges involved in translating this theory to practice. Key among these is the overhead of three way handshaking required for low duplication
forwarding under receiver diversity. To a degree, these approaches have been demonstrated by protocols such as ExOR [16]. Investigating the feasibility of receiver diversity deserves further study in systems.

6.2.4 Multichannel Operation

A number of exciting applications for wireless networks, have been challenged by low radio data rates (for examples in sensor networks, see [11], [48], [142]; for mesh networks, applications involving multiple multi-hop video transmissions face the same challenge). Even with carefully crafted hand-generated routes, congestion rapidly causes network collapse for networks of scale. Multi-channel MAC protocols offer a way to enhance the available bandwidth for such applications.

Two primary coordination challenges emerge: inter-node channel notification and radio reception blackout periods during transmission on a separate channel. Traditional shortest-path routing algorithms require a stable topology. This demands coordination of channel assignments and radio usage timings in order to leverage multi-channel network stacks efficiently. These are difficult problems, lending themselves to complicated solutions. The myopic, distributed routing decisions of backpressure algorithms appear well suited for networks with highly dynamic connectivity changes caused by multi-channel operation.

It is surprising, then, that multi-hop, multi-channel backpressure research has remained largely unexplored. We are particularly interested in formulations that may cleanly be implemented from a system standpoint. With single-channel radios dominating the wireless radio market, efforts to optimize channel selection requires coordination
that introduces further need for centralized or clustered arbitration. If channel switching is instead randomized, we release the burden of channel optimization, but still potentially gain in available network capacity. In settings where radio channel assignments evolve in an i.i.d. fashion, backpressure algorithms are provably throughput optimal. That is, the supported source rates over the dynamic connectivity graph formed by nodes randomly changing channels on time slot boundaries is greater than or equal to the source rates supported by any other algorithm.

Randomized channel switching is one area in which we feel there is potential for investigation. Backpressure algorithms, handling network dynamics well, should adapt to these randomized strategies much better than existing shortest-path based routing protocols. To what extent can randomized channel selection increase the maximum rate? We suggest evaluating the realistic gains of backpressure routing over randomized channel selection in single-channel radio systems.

There are challenges in implementing such a random-channel switching framework. Future work should investigate issues of network partitioning due to randomized channel selection, overheads associated with neighbor discovery upon channel switching, and optimizations in channel selection independent of the backpressure framework [15]. The neighbor discovery and forwarding process upon random channel change may leverage results from our proposed receiver diversity implementation. Translation will not be straight-forward, however, as the existing backpressure and receiver diversity work assumes that a node only broadcasts the head of line packet once the expected reward exceeds the penalty. This requires knowledge of neighbors and link reliabilities. One would need to investigate solutions to this information gap; potential solutions include
packet snoop-based learning algorithms to isolate neighbors and their backlogs, or short neighbor discovery broadcasts upon channel change.

6.2.5 Radio Sleep Scheduling

While we believe it is fairly straightforward to deploy backpressure stacks on synchronous sleep MAC (once all radios are awake, this is no different than typical operation), employing asynchronous sleep cycling under a backpressure framework presents some challenges. In order to play nice and avoid un-necessarily waking nodes, protocols designed for asynchronous MACs avoid the use of packet snooping and instead leverage explicit beacon packets for network metric dissemination (e.g. CTP [38]). By leveraging exponential backoffs in network metric beaconing, CTP drastically reduces the frequency with which nodes are woken out of their LPL sleep cycles. A backpressure stack, however, requires much more frequent updates from neighbor nodes. It has been proven that so long as neighbor backlog estimates are bounded in their error, the backpressure algorithm will still be throughput optimal but the time average bound on queue backlog is possibly enlarged [91]. Provided the very restricted packet queuing resources of wireless sensor network devices (BCP operates with a 12 packet forwarding queue) and the dynamic routing of backpressure stacks, lack of queue backlog feedback on a per-packet basis results in unacceptable fluctuations in queue backlogs (and hence queue drops). We believe that reasonably competitive aggregate duty cycling is possible over asynchronous MAC, provided the MAC is designed with backpressure applications in mind.

Specifically, we propose the modification of the LPL MAC such that the transmission pre-amble contains not only the destination address to which this packet is destined, but
also the backlog level of the transmitting node. This allows the overhearing radios to quickly simultaneously determine that they are not the destination while updating the neighbor backlog knowledge locally. Such a change should allow backpressure algorithms to run efficiently while asynchronous sleep is occurring.

6.2.6 Multicast and Any-to-Any Unicast Routing

Our research on backpressure routing has been restricted to the many to one convergecast traffic pattern for data collection in sensor networks. To extending this work to other kinds of applications and wireless networks, one needs to design and implement fully dynamic multicast and unicast routing. The core challenge in this extension is to do so in a scalable fashion, since the naive, most straightforward application of theoretical work on backpressure based dynamic unicast routing requires the maintenance of a separate queues for each flow (commodity).

The recent theoretical work discussed in Section 2.4.8.2 requires a substantially different operating paradigm from BCP. The changes greatly reduce the queue complexity and may provide comparable delay enhancements, though the delay bounds are still unknown. Future investigators might implement the shadow data and per-packet randomized routing techniques of Athanasopoulou et al. [9]. The challenges of such an implementation lie largely in link estimation. There are potential challenges to shadow data algorithms in real wireless sensor networks. Because BCP uses data-driven link estimation (actual packet transfers) to calculate link weights, the interference effects of the network are indirectly considered in routing decisions. If link weights are calculated over shadow data,
which is some $(1 + \epsilon)$ factor higher in rate than the actual flow, the data-driven link estimates will not accurately reflect the link statistics for the shadow data rate. This may cause unexpected behavior in the systems implementation of shadow data backpressure algorithms. In order to compute proper weights (and therefore scheduling), these algorithms may require modeling of the channel capacity, i.e. may not support data driven link estimation.

Assuming a shadow data based variant of BCP can be made to work, it would (in theory) be simple to extend into the multicast traffic domain (leveraging the recent piece of work by Bui et al. [19] discussed in Section 2.4.8.2).

An alternative to the shadow data approach might be to explore the combination of backpressure routing with geographic routing. Both backpressure and geographic routing share one feature in common - they both require only minimal decentralized exchange of information to enable per-packet forwarding decisions, instead of end-to-end route computation. Intuitively, the queue awareness advantage of backpressure should improve the efficiency of geographic routing in case of dynamics and heavy loading. More interestingly, we believe that the queue gradients established by backpressure routing may provide a very simple distributed solution to the problem of voids/holes that cause problems in sparse deployments. A key open theoretical question is whether the combined backpressure plus geographic routing is capable of supporting any-to-any traffic with low queue complexity.
6.2.7 Throughput Optimal MAC

In this work, our backpressure routing protocol runs atop the default CSMA MAC for TinyOS. This MAC is not known to be throughput optimal, nor constant-factor approximate. We surveyed recent work on throughput optimal MAC design in Section 2.4.8.4. Future investigation should determine whether, for lightweight networks such as wireless sensor networks, the techniques of Q-CSMA can be implemented with low overhead. We are concerned that even the three-way handshake required of receiver diversity techniques (above) may generate more overhead than benefit to network capacity.

Our initial results in the area of backpressure-derived MAC backoff for fixed routing indicated no throughput gains were found on wireless sensor networks [121]. Future investigators should verify this result in the backpressure dynamic routing setting, and for multiple throughput optimal MAC alternatives.

6.2.8 Rate Control Parameter Adaptation

In Chapter 5 we demonstrated the operating of source rate controllers within the backpressure stack. We evaluated performance for both proportional fair and alpha fair utility functions, and noted that both required careful parameter selection for optimal performance. While parameter selection in rate control protocols is often a function of topology, and is therefore not an unusual challenge, it is desirable to maintain the decentralized and highly agile capabilities demonstrated of the backpressure stack in Chapter 3. Substantial additional work remains in exploring solutions to this problem.
References


A. Jayasumana, N. Piratla, T. Banka, A. Bare, and R. Whitner. Improved packet reordering metrics. *IETF RFC 5236*.


Appendix: Proof of Delay Reduction Using LIFO Service Priority

We will now provide analysis for the LIFO delay advantage seen empirically in section 3.5.

**Definition 11.** Queue $Q_i(t)$ is defined as having a stabilized permanent backlog $b_i^\text{min} > 0$ if there exists a $t^*$ such that for all $t \geq t^*$, $Q_i(t) \geq b_i^\text{min}$ and there exists an infinite sequence of time slots $t^* \leq t_0 \leq t_1 \leq \cdots$ for which $Q_i(t_j) = b_i^\text{min}$.

That these stabilized permanent backlogs exist in BCP is evident, as the backlog at each node cannot undercut the penalty to reach the sink (the minimum cost path) scaled by $V$. This can be seen in the example of Figure 3.2. Recent theoretical work by Huang and Neely has shown that these stable backlog properties exist in more general backpressure systems [50].

**Definition 12.** The average delivered packet delay is defined as the average delay for packets passing through a queue but not trapped indefinitely within.

It is not useful to consider average delivered packet delay for arbitrary queueing systems, as without stability the volume of indefinitely trapped packets may grow unbounded. This metric becomes meaningful within the context of a stabilized permanent backlog queue operating with LIFO service priority. We can improve the average serviced packet delay by permanently trapping and effectively discarding $b_i^\text{min}$ packets.

**Theorem 5.** (The LIFO Delay Advantage for Constantly Backlogged Queues) Let $Q_i(t)$ be a queue with stabilized permanent backlog $b_i^\text{min}$ and arrival rate $\lambda_i$. Then the time average delivered packet delay relationship under FIFO ($W_{FIFO}$) and LIFO ($W_{LIFO}$) queueing disciplines is exactly:

$$W_{FIFO}^i = W_{LIFO}^i + \frac{b_i^\text{min}}{\lambda_i}$$

**Proof.** $Q_i(t)$ has stabilized permanent backlog $b_i^\text{min}$.

Case LIFO:
Under a LIFO discipline, any data arriving to find backlog greater than or equal to $b_i^\text{min}$ will be emptied infinitely often. The oldest $b_i^\text{min}$ packets within the LIFO queue at time $t^*$ are trapped indefinitely, and therefore are not considered in calculation of average delivered packet delay. Beyond time $t^*$, the average delivered packet delay of the LIFO queue is therefore equivalent to the average packet delay of a LIFO queue operating with the oldest $b_i^\text{min}$ packets removed. Let $N_{LIFO}^i$ be the time average number of packets in LIFO queue $i$ after removal of the $b_i^\text{min}$ trapped packets.

Case FIFO:
Under a FIFO discipline, the average delivered packet delay is equal to the average packet delay, as every arriving packet is eventually serviced. Let $N_{FIFO}^i$ be the time average number of packets in FIFO queue $i$. 

174
We can therefore relate the time average occupancies:

\[
N_{i}^{FIFO} = N_{i}^{LIFO} + b_{i}^{min} \implies \frac{N_{i}^{FIFO}}{\lambda_{i}} = \frac{N_{i}^{LIFO}}{\lambda_{i}} + \frac{b_{i}^{min}}{\lambda_{i}}
\]

\[
\implies W_{i}^{FIFO} = W_{i}^{LIFO} + \frac{b_{i}^{min}}{\lambda_{i}} \quad \text{(Little’s Applied Twice)}
\]

Where in the final step we use the fact that \(N_{i}^{FIFO}\) is serviced with FIFO service priority and that the modified LIFO queue empties infinitely often, therefore Little’s Theorem applies for both queues. \(\square\)