Routing Without Routes: The Backpressure Collection Protocol

Scott Moeller*, Avinash Sridharan*, Bhaskar Krishnamachari*, Omprakash Gnawali†
*University of Southern California, Los Angeles, CA
†Stanford University, Stanford, CA
{smoeller,asridhar,bkrishna}@usc.edu, gnawali@cs.stanford.edu

ABSTRACT

Current data collection protocols for wireless sensor networks are mostly based on quasi-static minimum-cost routing trees. We 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 a considerable theoretical literature on backpressure routing, it has not been implemented on practical systems to date due to concerns about packet looping, the effect of link losses, large packet delays, and scalability. Addressing these concerns, 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 highly dynamic network settings, including conditions of extreme external interference and highly mobile sinks.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*routing protocols, protocol verification*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*distributed networks, wireless communication*

General Terms

Algorithms, Design, Experimentation, Performance, Reliability, Verification

Keywords

Routing Protocol, Collection, Testbed Experiments, Stochastic Network Optimization, Wireless Sensor Networks

IPSN'10, April 12–16, 2010, Stockholm, Sweden.

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 [15]. 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 [3] 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 [6] 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 ([14], [20], [22], [24], [26], [27], [37]) referred to as *Utility Optimal Lyapunov Networking* algorithms in recent work by Neely [26][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.

[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

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 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) [15], 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 2 we give a more detailed description of backpressure routing. Section 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 4, and section 5 gives experimental evaluation. In section 6 we provide a brief survey of related systems and theory work. Finally, we conclude and discuss extensions in section 7.

2. BACKPRESSURE EXPLAINED

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.

when presenting related work in section 6.

2.1 Routing as a Stochastic Optimization Problem

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

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

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

$$\begin{aligned} \text{Minimize}: &\quad f(\bar{x}) \\ \text{Subject to}: &\quad \text{Strongly Stable} \end{aligned} \tag{2}$$

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 (2) using the Utility Optimal Lyapunov Networking framework ([14], [23], [24], [27]) 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:

$$w_{i,j} = (\Delta\mathcal{Q}_{i,j} - V \cdot \theta_{i,j}) \cdot \overline{\mathcal{R}}_{i,j} \tag{3}$$

Here, $\Delta\mathcal{Q}_{ij} = \mathcal{Q}_i - \mathcal{Q}_j$ is the queue differential (backpressure), with $\mathcal{Q}_i$ and $\mathcal{Q}_j$ representing the backlog of nodes i and j respectively, $\overline{\mathcal{R}}_{i,j}$ 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 (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 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.

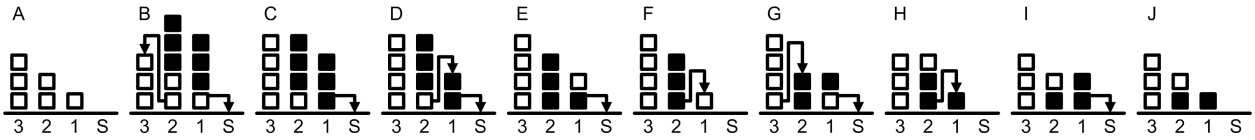


Figure 1: An intuitive example of backpressure routing on a four-node line network with FIFO queueing service. Three packets (in black) are injected at nodes 1 and 2 at time B, intended for the destination sink S.

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 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 $\theta_{i,j} = 1, V = 1, \overline{\mathcal{R}}_{i,j} = 1$.

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 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. NOVEL CONTRIBUTIONS OF BCP

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 (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.

3.1 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 [9, 40, 4, 15]. To incorporate ETX minimization into the backpressure framework, we use the following penalty function in the optimization problem (2):

$$f(\vec{x}) = \sum_{i} \sum_{j \in \mathcal{N}_i} x_{i,j}(t) \cdot \overline{ETX}_{i,j} \tag{4}$$

Where $\mathcal{N}_i$ is the set of neighbors of node i, $\overline{ETX}_{i,j}$ is the link ETX estimate, and $x_{ij}(t)$ is the forwarded packet count over link $i \rightarrow j$. Note that $f(\vec{x})$ satisfies the penalty function properties of problem (2), and yields a backpressure weight $w_{i,j}$ calculated by a node i for a given neighbor j which is the following:

$$w_{i,j} = (\Delta \mathcal{Q}_{i,j} - V \cdot \overline{ETX}_{i,j}) \cdot \overline{\mathcal{R}}_{i,j} \tag{5}$$

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 2 (i). As was observed in Figure 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.

3.2 Delay Reduction using LIFO

High source to sink delays are a well established problem in backpressure systems, resulting in significant recent theoretical focus ([5], [16], [25], [26], [42]). Under a FIFO service priority, data reaches the sink only when it is pushed through the chain of qeueus 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 [5]. 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

[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.

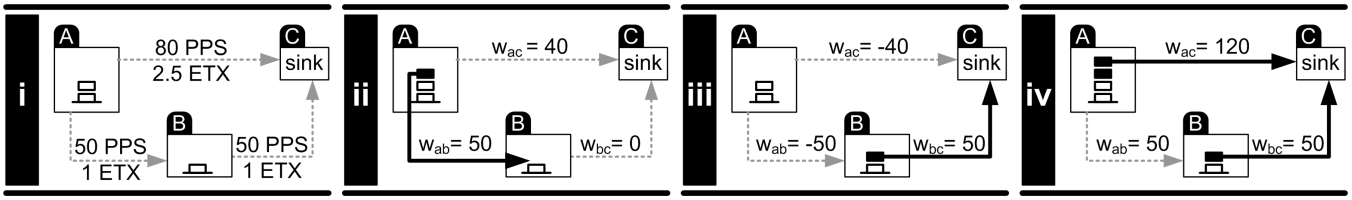


Figure 2: A three node network is given in (i), links are labeled with both rate and expected transmission count per packet. Bold links in (ii) through (iv) indicate links selected for packet forwarding. Weights are calculated using Equation (5) with $V = 1$.

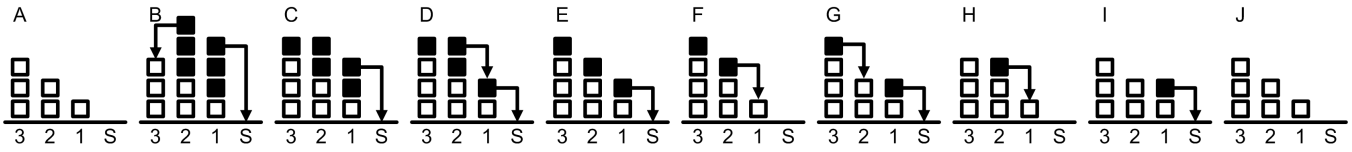


Figure 3: The four-node network of Figure 1, now with LIFO service priority. New additions to the queues flow over the existing gradient to the sink.

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. 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 Appendix A, we prove that for queue i with minimum recurrent backlog b_i^{min} and arrival rate λ_i, the average delay of a packet serviced by LIFO priority ($\overline{W}_i^{LIFO}$) is unboundedly lower than that achieved by FIFO priority ($\overline{W}_i^{FIFO}$):

$$\overline{W}_i^{FIFO} = \overline{W}_i^{LIFO} + \frac{b_i^{min}}{\lambda_i} \tag{6}$$

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 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 Scalability

Another substantial challenge faced by systems implementations of backpressure techniques is scalability. It can be clearly seen in Figure 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 5.2.2.

Recent theoretical work on queue stability [16] 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

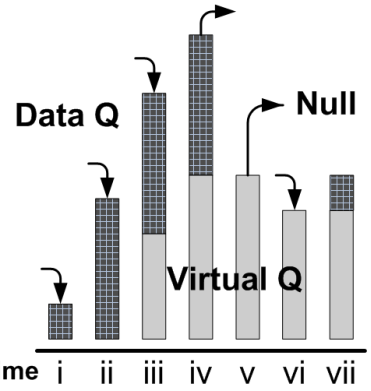


Figure 4: Our novel floating queues drop from the data queue during overflow, placing the discards within an underlying virtual queue. Services that cause data queue underflows generate null packets, reducing the virtual queue size.

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. 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 4. We show empirically in section 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.

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.

4.1 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 (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 τ, 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.

4.2 Weight Recalculation

The weight recalculation parameter τ, 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 4.1) 20 times a second in the event that no neighbor has a positive weight. We can use a larger τ in case weight re-computation keeps the CPU too busy for other tasks on the node.

4.3 Link Metric Estimation

Link metric estimation for transmissions $\overline{ETX}_{i\rightarrow j}$ and rate $\overline{\mathcal{R}}_{i\rightarrow j}$ 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 5 that the responsiveness of BCP to external interference is quite good with this parameter setting.

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. 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.

[4]BCP source code is publicly available online at http://anrg.usc.edu/∼scott/

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.

5. EXPERIMENTAL RESULTS

5.1 Experimental Methodology

We perform our evaluation experiments on motes labeled 1-40 in Tutornet [1], 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 [12], 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) [15], 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 ([38],[8],[11],[10],[7]). In our experimentation, CTP uses the 4 bit link estimator (4bitle [11]).

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 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 (5.3) and finally high sink mobility (5.4).

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 $\tau = 50$ ms[5]. Source rates vary from 0.25 to 1.66 packets per second.

5.2.1 Delay Performance

As discussed in section 3.2, delay in FIFO backpressure stacks actually grows with decreased loading, putting traditional backpressure at a severe disadvantage when compared to tree routing algorihtms. Figure 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.

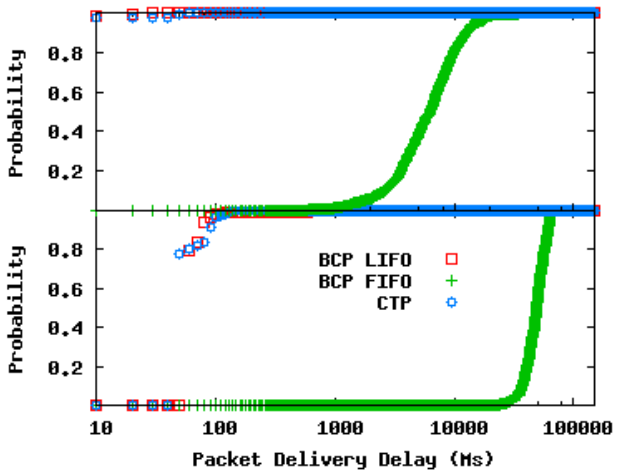


Figure 5: Source to sink delay CDF at 0.25 PPS for motes 4 and 40 under CTP, BCP-FIFO and BCP-LIFO.

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 [29]. Reordering density ([30], RFC 5236 [17]) 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.

[5]Due to space constraints, we do not provide a more detailed evaluation of the parameter settings for V and τ 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.

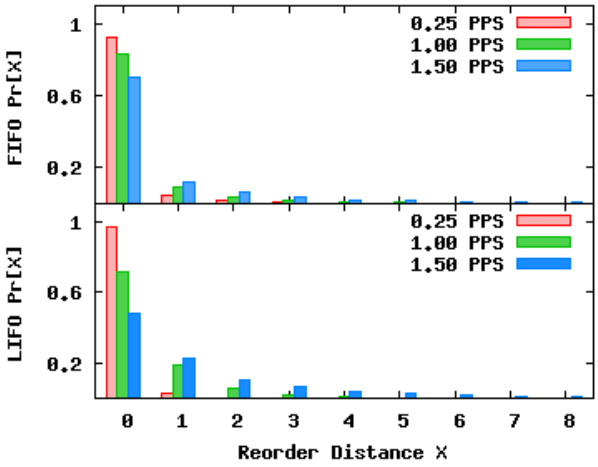


Figure 6: The Reordering Density for BCP under FIFO (top) and LIFO (bottom) servicing priorities for 0.25, 1.0 and 1.5 packets per second per source. The quasi-static tree routing mechanisms of CTP resulted in greater than 99.9% in-order delivery (Reordering $X = 0$) for 0.25 and 1.0 PPS tests.

Figure 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 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.

5.2.2 Scalability

In section 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 our 35 minute, 40 mote tests at 1 packet per second with BCP's floating queues disabled. Figure 7 gives per mote goodput, time average queue size and average source-

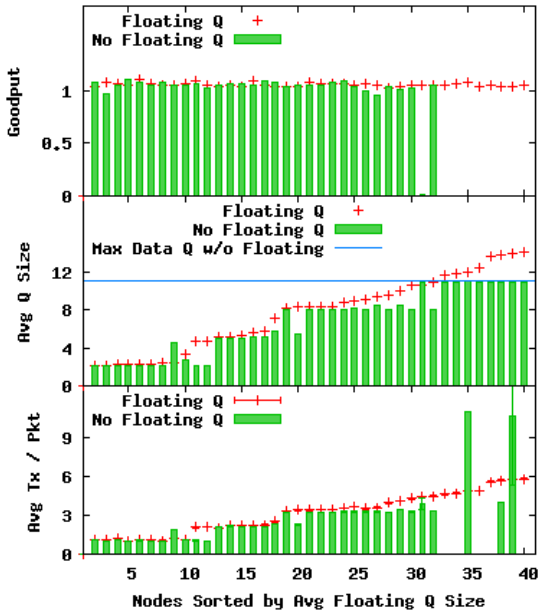


Figure 7: Comparison of BCP's per mote goodput, time average queue sizes and source-to-sink average packet transmissions per packet per source (with 95% confidence interval). Tests are run with and without BCP's floating queues disabled. The maximum data queue size is 11.

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 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 Tutornet.

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 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).

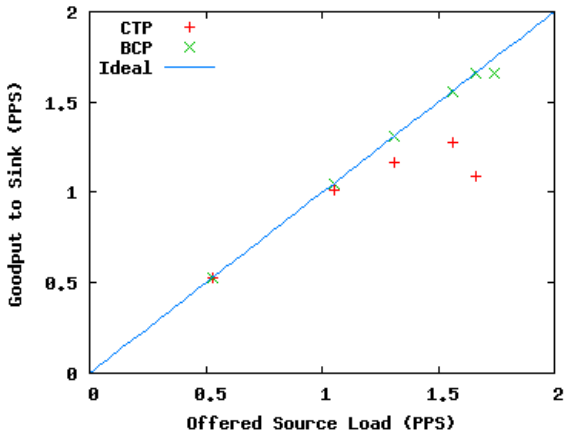


Figure 8: Goodput in static tests over source rate for BCP and CTP.

While avoiding hotspots, BCP should also be minimizing packet transmissions, as this is one of the dual goals of our underlying stochastic optimization. Figure 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).

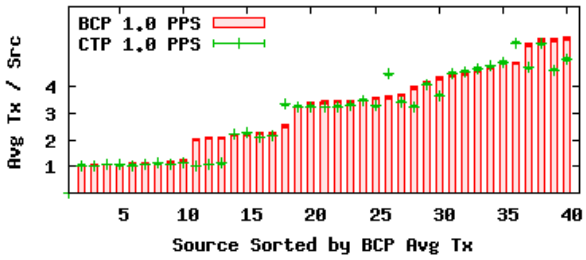


Figure 9: Average source-to-sink transmission count per packet per source (with 95% confidence interval) for the static 1.0 PPS experiment. Flow sources are sorted by average transmission count for BCP.

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.

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.

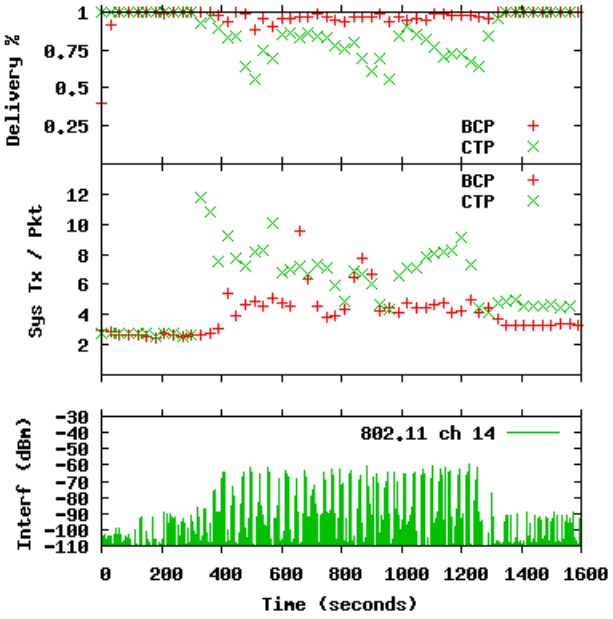


Figure 10: Thirty second windowed average sourced packet delivery ratio (top) and system transmissions per packet (middle). Spectrum analyzer results are plotted at bottom for the colliding 802.11 channel 14 traffic.

In the top portion of Figure 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 bottom portion of Figure 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.

5.4 Highly Mobile Sinks

There has been interest in the use of external mobile sinks within participatory sensing literature [6]. 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 moves every second, approximately the walking speed of a student, and cycles through the motes in 17 second before repeating. All 40 sources operate at 0.25 packets per second during the 35 minute test.

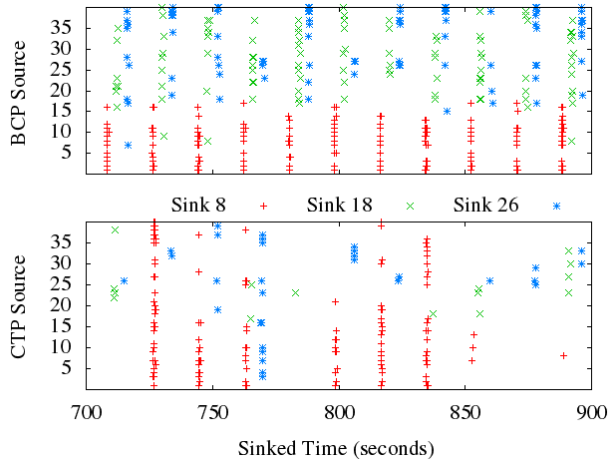
It is important to note that CTP was not designed for highly mobile sinks, but prior system implementations for mobile sinks (such as Hyper [33]) assume much lower sink mobility than modeled here (minutes, not seconds). We believe that performance results for CTP will not be unreasonable as approximations for Hyper and other tree-centric sink mobility solutions. Table 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 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.

	Mobility		Static	
	BCP	CTP	BCP	CTP
Delivery Ratio	0.996	0.590	0.969	.999
Average Tx/Packet	1.73	9.5	2.39	2.65

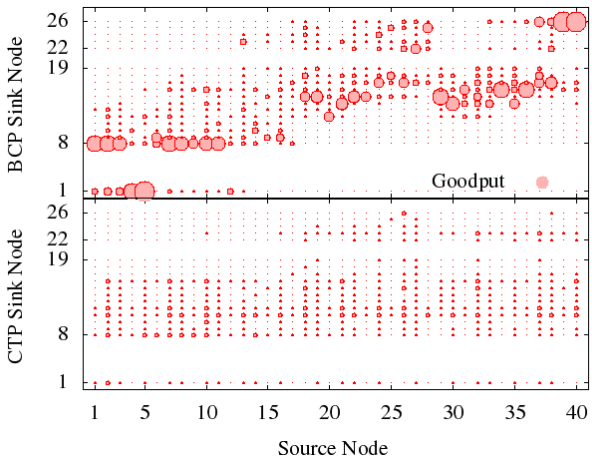
Table 1: Test results for highly mobile sink experiment at source rate 0.25 packets per second per source, provided alongside static network results from section 5.2.

In order to visualize packet deliveries over time, we plot sink timings per source for sinks 8, 18 and 26 in Figure 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 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.



(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 11: Comparison of performance of BCP and CTP under extreme sink mobility.

6. RELATED WORK

The intellectual roots of dynamic backpressure routing for multi-hop wireless networks lie in the seminal work by Tassiulas and Ephremides [37]. 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 ([14], [20], [22], [24], [26], [27]) 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 ([5], [21], [36]).

Researchers working on both optimization frameworks have lately worked to address delay reduction theoretically ([5], [16], [25], [26], [42]), but none stray from traditional FIFO assumptions. Our floating queue is similar in spirit to the virtual queues used in [16], 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 [2] and DiffQ [39], the queue differentials are used to change contention behavior at the MAC layer. In Horizon [32], 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 [2] and in the work by Sridharan *et al.* ([34],[35]). 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 [13], 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 [19], Horizon [32] and Arbutus [31] 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 are unable to do so at present due to lack of source code availability for the TinyOS platform.

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 [41] 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 τ). 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 ([2], [34], [35]). 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 [2], [39]). 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 ([18], [28]) to obtain bounds and guarantees on performance.

8. ACKNOWLEDGEMENTS

The authors thank Professor Michael Neely for numerous discussions on backpressure. This work was funded in part by NSF through grants CNS-0347621, CNS-0325875 and 0520235.

9. REFERENCES

[1] Tutornet. *http://enl.usc.edu/projects/tutornet/*.

[2] U. Akyol, M. Andrews, P. Gupta, and J. Hobby. Joint scheduling and congestion control in mobile ad-hoc networks. *IEEE INFOCOM*, Jan 2008.

[3] M. Bathula, M. Ramezanali, I. Pradhan, N. Patel, J. Gotschall, and N. Sridhar. A sensor network system for measuring traffic in short-term construction work zones. *DCOSS*, Jan 2009.

[4] S. Biswas and R. Morris. Exor: opportunistic multi-hop routing for wireless networks. *ACM SIGCOMM*, Jan 2005.

[5] L. Bui, R. Srikant, and A. Stolyar. Novel architectures and algorithms for delay reduction in back-pressure scheduling and routing. *ACM Infocom mini-conference*, 2008.

[6] J. Burke, D. Estrin, M. Hansen, and A. Parker. Participatory sensing. *World Sensor Web Workshop*, Jan 2006.

[7] J. I. Choi, M. A. Kazandjieva, M. Jain, and P. Levis. The case for a network protocol isolation layer. *Sensys*, Oct 2009.

[8] J. I. Choi, J. W. Lee, M. Wachs, and P. Levis. Opening the sensornet black box. *Proceedings of the International Workshop on Wireless Sensornet Architecture (WWSNA)*, Mar 2007.

[9] D. Couto, D. Aguayo, B. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. *ACM SIGCOMM*, Jan 2003.

[10] L. Filipponi, S. Santini, and A. Vitaletti. Data collection in wireless sensor networks for noise pollution monitoring. *DCOSS*, Jan 2008.

[11] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis. Four-bit wireless link estimation. *HotNets*, Oct 2007.

[12] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE*, Jan 2001.

[13] Y. Ganjali and N. McKeown. Routing in a highly dynamic topology. *IEEE SECON*, Jan 2005.

[14] L. Georgiadis, M. Neely, M. Neely, and L. Tassiulas. *Resource allocation and cross layer control in wireless networks*. 2006.

[15] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. *ACM Sensys*, Apr 2009.

[16] L. Huang and M. J. Neely. Delay reduction via lagrange multipliers in stochastic network optimization. *WiOpt*, Apr 2009.

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

[18] L. Jiang and J. Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *Proc. Allerton Conf. on Comm.*, Jan 2008.

[19] S. Katti and H. Balakrishnan. Symbol-level network coding for wireless mesh networks. *ACM SIGCOMM*, 2008.

[20] C. Li and M. Neely. Energy-optimal scheduling with dynamic channel acquisition in wireless downlinks. *IEEE CDC*, Jan 2007.

[21] J. Liu, A. Stolyar, M. Chiang, and H. Poor. Queue back-pressure random access in multi-hop wireless networks: Optimality and stability. *IEEE Trans on Information Theory*, Jan 2008.

[22] M. Neely. Order optimal delay for opportunistic scheduling in multi-user wireless uplinks and downlinks. *IEEE/ACM TON*, Jan 2008.

[23] M. Neely and R. Urgaonkar. Opportunism, backpressure, and stochastic optimization with the wireless broadcast advantage. *IEEE SSC*, Jan 2008.

[24] M. J. Neely. Dynamic power allocation and routing for satellite and wireless networks with time varying channels. *PhD Thesis, Massachusetts Institute of Technolog*, 2003.

[25] M. J. Neely. Super-fast delay tradeoffs for utility optimal fair scheduling in wireless networks. *IEEE JSAC*, Aug 2006.

[26] M. J. Neely. Intelligent packet dropping for optimal

energy-delay tradeoffs in wireless downlinks. *IEEE TAC*, Feb 2009.

[27] M. J. Neely, E. Modiano, and C.-P. Li. Fairness and optimal stochastic control for heterogeneous networks. *IEEE INFOCOM*, Sep 2005.

[28] J. Ni and R. Srikant. Q-csma: Queue-length based csma/ca algorithms for achieving maximum throughput and low delay in wireless networks. *Proc. of Information Theory and Applications Workshop*, Jan 2009.

[29] N. Piratla and A. Jayasumana. Reordering of packets due to multipath forwarding-an analysis. *IEEE ICC*, Jan 2006.

[30] N. Piratla and A. Jayasumana. Metrics for packet reordering-a comparative analysis. *International Journal of Communication Systems*, Jan 2008.

[31] D. Puccinelli and M. Haenggi. Reliable data delivery in large-scale low-power sensor networks. *ACM TOSN*, Sep 2009.

[32] B. Radunovic, C. Gkantsidis, and D. Gunawardena. Horizon: Balancing tcp over multiple paths in wireless mesh network. *ACM MOBICOM*, Jan 2008.

[33] T. Schoellhammer, B. Greenstein, and D. Estrin. Hyper: A routing protocol to support mobile users of sensor networks. *Tech Report 2013, CENS*, Oct 2006.

[34] A. Sridharan, S. Moeller, and B. Krishnamachari. Implementing backpressure-based rate control in wireless networks. *ITA Workshop*, Sep 2008.

[35] A. Sridharan, S. Moeller, and B. Krishnamachari. Making distributed rate control using lyapunov drifts a reality in wireless sensor networks. *WiOpt*, 2008.

[36] A. Stolyar. Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm. *Queueing Systems*, Jan 2005.

[37] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and schedulingpolicies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 1992.

[38] M. Wachs, J. I. Choi, J. W. Lee, K. Srinivasan, Z. Chen, M. Jain, and P. Levis. Visibility: A new metric for protocol design. *ACM Sensys*, Sep 2007.

[39] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee. Diffq: Practical differential backlog congestion control for wireless networks. *INFOCOM*, Jan 2009.

[40] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. *ACM Sensys*, 2003.

[41] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. *IEEE INFOCOM*, Jan 2002.

[42] L. Ying, S. Shakkottai, and A. Reddy. On combining shortest-path and back-pressure routing over multihop wireless networks. *Proceedings of the IEEE INFOCOM*, Jan 2009.

APPENDIX

A. PROOF OF DELAY REDUCTION USING LIFO SERVICE PRIORITY

We will now provide analysis for the LIFO delay advantage seen empirically in section 5.

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

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 2. Recent theoretical work by Huang and Neely has shown that these stable backlog properties exist in more general backpressure systems [16].

Definition 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^{min} *packets.*

THEOREM A.1. *(The LIFO Delay Advantage for Constantly Backlogged Queues) Let* $\mathcal{Q}_i(t)$ *be a queue with stabilized permanent backlog* b_i^{min} *and arrival rate* λ_i*. Then the time average delivered packet delay relationship under FIFO (*$\overline{W}_{FIFO}$*) and LIFO (*$\overline{W}_{LIFO}$*) queueing disciplines is exactly:*

$$\overline{W}_i^{FIFO} = \overline{W}_i^{LIFO} + \frac{b_i^{min}}{\lambda_i}$$

PROOF. $\mathcal{Q}_i(t)$ *has stabilized permanent backlog* b_i^{min}*.*

Case LIFO:

Under a LIFO discipline, any data arriving to find backlog greater than or equal to b_i^{min} *will be emptied infinitely often. The oldest* b_i^{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^{min} *packets removed. Let* $\overline{N}_i^{LIFO}$ *be the time average number of packets in LIFO queue* i *after removal of the* b_i^{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 $\overline{N}_i^{FIFO}$ *be the time average number of packets in FIFO queue* i*.*

We can therefore relate the time average occupancies:

$$\overline{N}_i^{FIFO} = \overline{N}_i^{LIFO} + b_i^{min} \implies \frac{\overline{N}_i^{FIFO}}{\lambda_i} = \frac{\overline{N}_i^{LIFO}}{\lambda_i} + \frac{b_i^{min}}{\lambda_i}$$
$$\implies \overline{W}_i^{FIFO} = \overline{W}_i^{LIFO} + \frac{b_i^{min}}{\lambda_i}$$
(Little's Applied Twice)

Where in the final step we use the fact that $\overline{N}_i^{FIFO}$ is serviced with FIFO service priorty and that the modified LIFO queue empties infinitely often, therefore Little's Theorem applies for both queues. □