RCRT: Rate-Controlled Reliable Transport Protocol for Wireless Sensor Networks

JEONGYEUP PAEK
University of Southern California
and
RAMESH GOVINDAN
University of Southern California

Emerging high-rate applications (imaging, structural monitoring, acoustic localization) will need to transport large volumes of data concurrently from several sensors. These applications are also loss-intolerant. A key requirement for such applications, then, is a protocol that reliably transport sensor data from many sources to one or more sinks without incurring congestion collapse. In this paper, we discuss RCRT, a rate-controlled reliable transport protocol suitable for constrained sensor nodes. RCRT uses end-to-end explicit loss recovery, but places all the congestion detection and rate adaptation functionality in the sinks. This has two important advantages: efficiency and flexibility. Because sinks make rate allocation decisions, they are able to achieve greater efficiency since they have a more comprehensive view of network behavior. For the same reason, it is possible to alter the rate allocation decisions (for example, from one that ensures that all nodes get the same rate, to one that ensures that nodes get rates in proportion to their demands), without modifying sensor code at all. We evaluate RCRT extensively on a 40-node wireless sensor network testbed and show that RCRT achieves 1.7 times the rate achieved by IFRC and 1.4 times that of WRCP, two recently proposed interference-aware distributed rate-control protocols. We also present results from a 3-month-long 19-node real world deployment of RCRT in an imaging application and show that RCRT works well in real long-term deployments. [1]

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Wireless communication*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Algorithm, Design, Experimentation, Performance

Author's address: University of Southern California, Computer Science Department, Embedded Networks Laboratory, Ronald Tutor Hall (RTH) 418, 3710 S. McClintock Avenue, Los Angeles, CA 90089; email:{jpeak, ramesh}@usc.edu

[1] **An earlier version of this paper appeared in the Proceedings of the ACM SenSys 2007. This manuscript differs from that published version in many ways: it discusses experiences and results from a 3-month-long moderate-size real-world deployment (Section 5), discusses more design details (Section 3), includes larger-scale experiments as well as experiments on networks with different density and depth (Section 4.3), compares against another protocol (Section 4.2.5), investigates the effect of additive increase parameter (Section 4.5), and reports the sensitivity of the protocol to parameters in other layers of a networking system (e.g. link-layer retransmissions, routing layer forwarding queue size) (Section 4.4).**

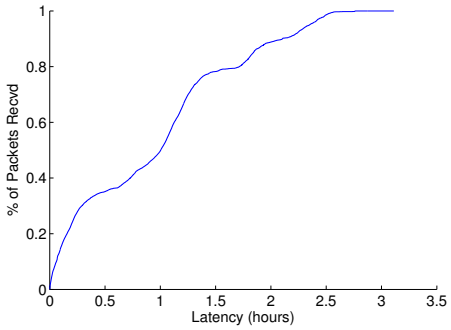

Fig. 1. CDF plot of the packet latency plot from Wisden deployment, 2004. Reception time on the x-axis, percentage on the y-axis.

Additional Key Words and Phrases: Sensor networks, Congestion control, Reliable, Transport protocol, Centralized, Tiered network

1. INTRODUCTION

As sensor network software and hardware matures, it is becoming increasingly possible to conceive of a class of applications that has received relatively little attention so far — applications requiring the transport of high-rate data. Sources for high-rate data include imagers, microphones, and accelerometers. These sensors in turn motivate several interesting applications in surveillance, precision agriculture, structural damage assessment, and military target tracking.

To support these emerging applications, we need to solve two problems. First, wireless sensors have limited radio bandwidth. A collection of sensors generating high-rate data can easily overwhelm the network to the point of congestion collapse, where the network is unable to perform useful work because its capacity is exceeded. Second, applications that use high-rate sensors of the kind described above are often loss-intolerant. For example, source localization algorithms [Ali et al. 2007] use time difference of arrival between comparable samples at different nodes, and structural monitoring algorithms estimate structural mode shapes [Caffrey et al. 2004] by correlating comparable samples observed at different nodes. Even if application data can be compressed to reduce bandwidth requirements [Hicks et al. 2008], compression algorithms require reliable delivery.

But a protocol for end-to-end reliable delivery of voluminous data can, if not carefully designed, result in poor network performance. We learned this lesson in a dramatic way during our deployment of wireless sensors on the Four Seasons building in Los Angeles about 4 years ago [Paek et al. 2005]. In this deployment, sensor nodes measured vibrations and transmitted it to a central node, over multiple hops, at a pre-configured fixed rate. During our deployment, severe network congestion occurred, a behavior we did not observe during extensive pre-deployment tests. Figure 1 plots the CDF of the packet latencies from the entire experiments for all nodes. Greater than half the packets were delivered more than an hour after they were first injected into the network!

While many sensor network transport protocols have been studied in the literature, most of them solve one of the two problems identified above (Section 2): they either provide reliable end-to-end delivery of data from every sensor to a sink, or discuss a congestion control mechanism without ensuring end-to-end reliable delivery.

In this paper, we discuss the design and implementation of a transport protocol that ensures reliable delivery of sensor data from a collection of sensors to a base station, while avoiding congestion collapse. However, we place two other requirements on the design of this transport protocol. First, unlike most existing proposals which, implicitly or explicitly, support only a single stream of sensor data from each network node, we require the network to be able to support multiple concurrent streams from each sensor node. We foresee that future sensor network deployments will be multi-user systems, with concurrently executing applications. Second, while much existing work has assumed a specific way to allocate network capacity to all sensors (*e.g.*, a fair allocation), we require our solution to separate the capacity allocation policy from the underlying transport mechanisms. It is unclear, yet, if there exists a single traffic allocation policy that would satisfy the needs of all sensor network applications.

Our solution, RCRT, has many different components, many of which are novel (Section 3). It uses relatively standard mechanisms for end-to-end reliable delivery; the base station (or *sink*) discovers missing packets and explicitly requests them from the sensors. However, its congestion control functionality, in a significant departure from much of the prior work, is implemented in the sink. The sink has a comprehensive view of the performance of the network, and it uses this perspective to control traffic allocation in a more efficient way than would be possible with decentralized congestion control. RCRT employs a novel congestion detection technique, in which the sink decides that the network is congested if the time to repair a loss is significantly higher than a round-trip time. Moreover, it de-couples rate adaptation from rate allocation; that is, the RCRT sink first decides how much the total traffic needs to be reduced (or increased) in response to congestion (or lack thereof), then separately decides how to allocate the increase or decrease to different sources. This decoupling allows a network administrator to assign different capacity allocation policies for different applications.

We have implemented RCRT's sink-side functionality on a PC-class platform (our code ports to embedded systems such as Stargates as well), and the sensor-side functionality on the Telosb, MicaZ, and Mica2 mote platforms. Our detailed evaluation (Section 4) of RCRT performance brings out many of its features: its ability to dynamically respond to congestion, its flexibility, robustness, and its support for multiple applications. More important, our evaluations show that RCRT is able to achieve 1.7 times the network throughput of IFRC [Rangwala et al. 2006] and 1.4 times that of WRCP [Sridharan and Krishnamachari 2009], two recently proposed approach that implements decentralized congestion control but does not guarantee end-to-end reliability. RCRT is able to achieve this because its traffic control algorithms are able to better estimate and control the network capacity given the sink's comprehensive perspective into network performance. Finally, we have employed RCRT in a 3-month-long moderate-sized real-world deployment of

	Distributed Congestion Control	*Centralized Congestion Control*	*No Congestion Control*
Reliable	Flush, STCP, Hop	**RCRT**	Wisden, Tenet RMST
Unreliable	WRCP, IFRC, Fusion, CODA	QCRA, ESRT	Surge, CTP, RBC, CentRoute, Koala

Table I. Sensor Network Transport Protocols: A Taxonomy

an image-based environmental monitoring application, in which RCRT have collected 83 million packets with 100% reliability, and show that RCRT is capable of running robustly and efficiently in long term real deployments.

2. RELATED WORK

To place our work in context, we taxonomize sensor network transport protocols as shown in Table I. We distinguish transport protocols by whether they provide end-to-end reliability or not, whether they implement congestion control or not, and if they do, whether the congestion control implementation is distributed or centralized (at a sink, for example). As Table I shows, RCRT is, to the best of our knowledge, the only instance of a reliable transport protocol that implements congestion control in a centralized manner. Furthermore, to our knowledge, although there is a large literature on congestion control in wired and wireless networks, this specific problem has not been addressed in those contexts. However, many of our individual design decisions draw from that literature; we cite the relevant pieces of work when we describe the detailed design of RCRT in Section 3. We now discuss each element of Table I in turn.

The simplest transport protocols are those that do not guarantee end-to-end reliability, and implement no congestion control. Surge (TinyOS) and CTP [Gnawali et al. 2009] can be thought of as implementing such a simple transport protocol. CentRoute/DataRel [Stathopoulos et al. 2005] centrally computes efficient source routes to individual motes on demand and provides a TCP-like abstraction for transporting data from a mote to a nearest gateway in a tiered network. It implements a fixed number of end-to-end retransmissions to improve reliability, but does not incorporate congestion control. RBC [Zhang et al. 2003] is a hop-by-hop reliable transport scheme optimized for real-time many-to-one delivery of bursty event data, and uses retransmission scheduling to increase reliability and reduce latency. Koala [Musaloiu-E. et al. 2008] is a data retrieval system optimized for bulk data delivery at low duty cycles. In Koala, the gateway initiates bulk delivery by waking up and installing routing paths on the motes dynamically. However, neither RBC nor Koala were designed for continuous flows, and they do not guarantee end-to-end reliability.

Next come the class of transport protocols that provide end-to-end reliability, but implement no congestion control. RMST (Reliable Multi-Segment Transport) [Stann and Heidemann 2003] and the transport protocol implemented in Wisden [Xu et al. 2004] and Tenet [Paek et al. 2010] are examples of such protocols. RMST is a hop-by-hop reliable transport protocol built on top of Directed Diffusion [Intanagonwiwat et al. 2002] in which loss is repaired hop-by-hop using caches

in the intermediate nodes. RMST guarantees reliability, but it is designed for larger and more capable platforms. Wisden's transport protocol (Tenet's is very similar) provides end-to-end reliability of sensor data transmitted from a field of sensors to a single sink. However, in both these systems, the rate at which this data is transmitted by a node must be manually set by a system administrator.

Some research has examined centralized congestion control without end-to-end reliability guarantees. QCRA [Bian et al. 2007] (Quasi-static Centralized Rate Allocation) is a centralized rate allocation scheme that tries to achieve fair and near-optimal rate allocation for each node given the topology, routing tree, and link loss rate information. ESRT (Event-to-Sink Reliable Transport) [Sankarasubramaniam et al. 2003], is also a centralized rate control scheme for event-driven applications where the base station requests all source nodes to increase or decrease rate in order to achieve the desired event reliability. But ESRT assumes that the sink can communicate with all sources in one hop, and has only been evaluated in simulation.

There is a body of work that has designed distributed congestion control schemes without regard to end-to-end reliability. IFRC (Interference-aware Fair Rate Control) [Rangwala et al. 2006] is a distributed rate allocation scheme that uses queue size to detect congestion, shares the congestion state through overhearing, and converges to fair and efficient rates for each node. Even though IFRC does not provide end-to-end reliability, it is closest in spirit to our work in that it attempts to find a fair and efficient transmission rate that avoids congestion collapse. We quantitatively compare IFRC and RCRT in a later section, but note that a) RCRT has greater flexibility than IFRC since many different traffic allocation policies can be implemented in the RCRT sink without changing any code at the sensor nodes, and b) RCRT does not require sophisticated parameter tuning for stability as IFRC does. WRCP (Wireless Rate Control Protocol) [Sridharan and Krishnamachari 2009] is another recently proposed distributed rate control scheme that uses an estimate of receiver capacity and estimates of active flow counts to allocate fair rates to each node. It does not provide end-to-end reliability, and RCRT is more flexible and requires less parameter tuning than WRCP as well. Although WRCP converges faster to a less oscillatory rate allocation than IFRC, RCRT achieves better goodput than WRCP. Also different from RCRT is work on congestion mitigation: Fusion [Hull et al. 2004] uses hop-by-hop congestion control and CODA (Congestion Detection and Avoidance) [Wan et al. 2003] uses end-to-end flow control along with hop-by-hop back-pressure for this purpose.

Finally, some work has examined distributed congestion control of end-to-end reliable transport. STCP [Iyer et al. 2005] proposes to use RED [Floyd and Jacobson 1993]-style congestion detection in sensor nodes and a slightly modified form of TCP end-to-end. To our knowledge, STCP has not been evaluated on a real wireless testbed. By contrast, Flush [Kim et al. 2007] is a reliable transport protocol designed for large diameter sensor networks. Unlike in RCRT, at any instant at most one node can transport its data to the sink using Flush. Also, Hop [Li et al. 2009] is a recently proposed wireless transport protocol that uses reliable per-hop block transfer and backpressure flow control. It eliminates the overhead and latency involved in end-to-end control, but it is designed for larger and more capable platforms with sufficient memory.

Not included in the taxonomy described in Figure I are protocols for reliable dissemination. PSFQ [Wan et al. 2002], Deluge [Hui and Culler 2004], and TRD [Paek et al. 2010] are reliable dissemination protocols designed for reprogramming or retasking the sensor network from a base station, and not for transport of data in the other direction. Also omitted from Figure I is prior work on congestion sharing in wired networks (for example, Ensemble-TCP [Eggert et al. 2000], and Integrated Congestion Management Architecture [Balakrishnan et al. 1999]). These proposals, though not directly applicable to sensor networks, bear some resemblance to RCRT's centralized congestion control.

3. RATE CONTROLLED RELIABLE TRANSPORT

In this section, we start by describing the goals of RCRT and discuss how its overall design meets those goals. We then delve into the details of individual components of RCRT: end-to-end reliability, congestion detection, rate adaptation and rate allocation. We conclude with a discussion of RCRT's limitations.

3.1 Design Goals

RCRT is designed for sensor networks in which sensor readings are transmitted from one or more sensors (*sources*) to a base station (or *sink*). It is also applicable to tiered sensor networks [Paek et al. 2010], where sensors may be transporting sensor readings to the nearest gateway (master, in the terminology of [Paek et al. 2010]), which in turn routes the messages to a designated upper-tier node (which we call the *sink*). RCRT does *not* require all sensors to be transmitting data. More generally, sensors may start and end data transmissions at arbitrary times that are not known *a priori*.

Six goals guide the design of RCRT. They are:

Reliable end-to-end transmission. Our first goal is to achieve complete end-to-end reliability of all data transmitted by each sensor to a sink. Of course, this is only possible if the network is not partitioned: that is, for each source, there exists a network level path to the sink where each link has a non-zero packet reception rate. This goal is motivated by emerging high data rate applications which are loss-intolerant; examples of these applications include networked imaging [Rahimi et al. 2005], acoustic source localization [Ali et al. 2007], and structural health monitoring [Chintalapudi et al. 2006]. In each of these cases, processing algorithms are extremely sensitive to packet loss, and they are rarely interested in inter-node data aggregation. For example, source localization algorithms use time difference of arrival between comparable samples at different nodes, and structural monitoring algorithms estimate structural mode shapes by correlating comparable samples observed at different nodes. In either case, the loss of samples can adversely affect the accuracy of the algorithm.

Network Efficiency. Our second goal is to maintain the network at an efficient operating point. Specifically, we wish to avoid *congestion collapse* [Floyd 2000], a regime in which sources are sending data faster than the network can transport them to the base station. In this regime, no useful work gets done by the network, since packets are repeatedly lost and continually retransmitted. In addition, we wish to ensure that sources transmit their sensor readings to the base station at as high a rate as possible. Since sources may start transmitting sensor readings at

arbitrary times, this rate cannot be determined *a priori*, but must be adaptively discovered.

Support for concurrent applications. While much prior work on sensor network transport has focused on supporting a single sensing application, we wish to explicitly support multiple concurrent applications in RCRT. For example, a user might want to run a network diagnostic application while a sensing application is running, or run an actuation application depending on the results reported from an environmental sensing application. Future sensor network deployments are likely to evolve to being multi-user or multi-application systems, so it is important to consider this design criterion at the outset.

Flexibility. Another goal is to allow different applications to choose different *capacity allocation policies*. A capacity allocation policy determines how the overall network capacity is divided up among the different sources. In some homogeneous deployments, where all the sensors are generating data at exactly the same rates, it may be necessary to divide up the capacity equally (in a *fair* manner). In other cases, some sources might need a proportionally larger allocation since, for example, they might be transmitting images. An important sub-goal is that this flexibility should not come at the expense of requiring code modifications on the sensors; this is clearly desirable. This goal distinguishes our work from distributed congestion control mechanisms that implicitly embed a traffic allocation policy within the network. For example, IFRC can only support fair and weighted fair allocations [Rangwala et al. 2006].

Minimal Sensor Functionality. We wish to keep as much of the protocol functionality out of the sensors as possible. This goal is motivated by the constraints of the current generation of sensors. More generally, however, it is motivated by the observation that, for our problem, it is possible to achieve overall system simplicity by moving as much of the intelligence as possible out of the sensor network and into the sink.

Robustness. Finally, we require that RCRT be robust to routing dynamics and to nodes entering and leaving the system. This implies that traffic allocations to sensors can dynamically change, as can the locations of congested nodes. The system must be able to dynamically adapt to these changes.

3.2 RCRT Overview

To describe RCRT, we introduce the following notation and terminology. We define a *sink* as an entity (software program, or part thereof) which runs on a base station (or an upper-tier node in a tiered network) and which collects data from one or more sensors (sources). Let $\mathbb{S}$ be the set of sensor nodes that have data to send to a sink, and $\mathbb{K}$ the set of all sinks in the network. RCRT is oblivious to the kind of data sourced by the sensors: they can be raw samples, processed time series, images, and so forth. More than one sink can be running concurrently in the sensor network. We use the notation $f_{i,j}$ to denote the *flow* of data from source $i \in \mathbb{S}$ for sink $j \in \mathbb{K}$. Of course, this flow is delivered from the source over (possibly) multiple hops to the base station. A sensor i may source several flows $f_{i,j}$ for different sinks j. Finally, each sink j is associated with a capacity allocation policy P_j which determines how network capacity is divided up across flows $f_{i,j}$ for $\forall i \in \mathbb{S}$. The simplest P_j is one in which each flow $f_{i,j}$ gets an equal share of the network capacity

Function	Where	How
End-to-end Loss Recovery	Source and Sink	End-to-end NACKs
Congestion Detection	Sink	Based on time to recover loss
Rate Adaptation	Sink	Based on total traffic, with additive increase and decrease based on loss rate
Rate Allocation	Sink	Based on application-specified capacity allocation policy

Table II. RCRT Components

(a *fair* allocation). We give more examples of P_j later.

RCRT provides reliable, sequenced delivery of flows $f_{i,j}$. Furthermore, RCRT ensures that, for a given application j, the available network capacity is allocated to each flow according to policy P_j. Specifically, each flow $f_{i,j}$ is allocated a rate $r_{i,j}(t)$ at each instant t that is in accordance with policy P_j. Thus, for a fair allocation policy, all sensors would receive equal $r_{i,j}(t)$.

How does RCRT achieve all this? Table II describes the various components of RCRT. End-to-end reliability is achieved using end-to-end negative acknowledgments. A particularly novel aspect of RCRT is that its *traffic management functionality resides at the sink*. Specifically, each sink determines congestion levels and makes rate allocation decisions. Once sink j decides the rate $r_{i,j}$, it either piggybacks this rate on a negative acknowledgment packet, or sends a separate feedback packet, to source i. The source nodes simply react to the feedback provided by the sink.

At the sink, RCRT has three distinct logical components. The *congestion detection* component observes the packet loss and recovery dynamics (which packets have been lost, how long it takes to recover a loss) across every flow $f_{i,j}$, and decides if the network is congested. Once it determines that the network is congested, the *rate adaptation* component estimates the total sustainable traffic $R(t)$ in the network. Then, the *rate allocation* component decreases the flow rates $r_{i,j}(t)$ to achieve $R(t)$, while conforming to policy P_j. Conversely, when the network is not congested, the rate adaptation component additively increases the overall rate $R(t)$, and the rate allocation component determines $r_{i,j}(t)$. In our current implementation, multiple sinks do not cooperate with each other to determine congestion, adapt or allocate rates; doing so is likely to provide higher efficiency gains and a more equitable rate allocation, and we have left this to future work.

RCRT satisfies our design goals (Section 3.1). By design, it provides end-to-end reliability and attempts to keep the network operating efficiently while supporting multiple applications, each with its own capacity allocation policy. Much of the traffic management functionality in RCRT is centralized at the sink, keeping the sensors as simple as possible.

Finally, our assumptions about the link layer and the routing layer are largely consistent with current practice. Although our experiments are conducted using the default CSMA MAC layer in TinyOS, the design of RCRT does not depend on any features specific to a particular MAC layer. Link-level retransmissions at

the MAC layer can improve the performance of RCRT but are not essential for its correctness. We discuss this in more detail in Section 3.8, and verify our argument through evaluation in Section 4.4.

We assume that the sensor nodes run a routing protocol that dynamically selects a path from each node to the sink and also a reverse path from the sink to each node. Also, RCRT does not assume symmetric routing between a source and the sink. Our experiments (Section 4) are conducted using TinyOS's tree-based routing protocol, MultiHopLQI, but it can work with other routing protocols such as CentRoute [Stathopoulos et al. 2005] which satisfy these requirements.

Finally, RCRT focuses on the transport protocol itself. Cross-layer techniques for congestion control that exploit radio, MAC or routing protocol features are beyond the scope of this work.

In the following sections, we discuss the detailed design of RCRT. As described above, RCRT sinks act independently in rate allocation decisions. So, in what follows, we drop the subscript j, with the understanding that all the behavior described below refers to a sink and its associated flows.

3.3 End-to-end Reliability

Unlike TCP [Jacobson 1988], RCRT implements a NACK-based end-to-end loss recovery scheme to guarantee 100% reliable data delivery. The sink detects packet losses and repairs them by requesting end-to-end retransmissions from source nodes. This design leverages the fact that the base station has significantly more memory and can keep track of all missing packets. Each sensor node stores a copy of every data packet that it originates in its local retransmit buffer when transmitting the packet to the sink. The sink keeps track of sequence numbers of packets that it receives on each flow. A gap in the sequence number of received packets indicates packet loss. The sink maintains a list of missing packets per flow. When losses are detected, the sequence numbers of the lost packets are inserted into this list. Entries in this list of missing packets are sent as negative acknowledgments (NACKs) by the sink to each source. The use of NACKs avoids an "ACK implosion" [Floyd et al. 1997], where the acknowledgments of successful receptions sent by the sink overwhelm the network. Upon receiving a NACK, the source retransmits the requested packets to repair the losses. Also, the source determines when it can safely overwrite packets in the retransmit buffer by looking at the cumulative ACK sequence number piggybacked in all feedback packets (Section 3.7 describes cumulative ACKs and feedback packets).

The sink also maintains a list of out-of-order packets for each flow to provide in-order delivery of data packets to the application. This list contains packets that are received at the sink but have not been passed to the application layer because there are one or more gaps in the sequence numbers. For example, if sequence numbers $[1, 2, 3, 5, 6]$ have been received so far, packets 5 and 6 are inserted into the out-of-order packets list. When packet 4 is received, the sink passes packets $4, 5, 6$ to the application and removes $5, 6$ from the out-of-order packets list.

RCRT uses the per-flow lists of missing and out-of-order packets for detecting congestion and adapting rates, as we shall discuss below.

3.4 Congestion Detection

An important technical challenge for RCRT is the design of a mechanism for congestion detection. Various techniques have been proposed in the wireless and sensor network literature to measure the level of congestion *at a node*. Broadly speaking, these techniques either directly measure the channel utilization around a node [Wan et al. 2003], or measure the queue occupancy at the node [Hull et al. 2004; Rangwala et al. 2006]. These techniques are similar in spirit to approaches that attempt to detect incipient congestion in Internet routers [Floyd and Jacobson 1993; Katabi et al. 2002]. By contrast, RCRT attempts to measure, at the sink, whether *the network is congested*. This approach is closer in spirit to approaches in wired networks that have attempted to detect congestion at the ends [Jacobson 1988; Ramakrishnan and Jain 1990], but with one important difference: in RCRT, a sink has a more extensive view of network performance than, for example, a TCP sender, since the sink receives traffic from many sources.

However, it would be incorrect to merely borrow congestion detection methods used in wired networks. For example, TCP uses a single packet drop to infer that the network is congested. In a wireless network, it is well known that such a congestion detection mechanism can result in extremely poor transport performance because wireless links tend to exhibit relatively high packet loss rates.

Accordingly, RCRT bases its congestion detection mechanism on a completely different approach. This approach is in line with RCRT's primary goal of providing end-to-end reliability. Its congestion detection mechanism is based on the following intuition, derived from our early experimental deployment and discussed in Figure 1: that the network is uncongested as long as end-to-end losses are repaired *"quickly enough"*. This intuition permits a few end-to-end losses caused by transient congestion, or by poor wireless links. Furthermore, it permits the sources to transmit at a higher rate even if there are occasional end-to-end losses, since those losses can be recovered by the mechanism described in the previous section. For this reason, RCRT uses the *"time to recover loss"* as a congestion indicator.

Recall that RCRT maintains a per-flow list of packets that have been received out of order, and also a per-flow list of packets that are missing (Section 3.3). Now, the sum of the length of these two lists minus 1 indicates how many packets should have been received *after* the first unrecovered packet loss, which reflects how much time has passed since the first unrecovered loss. Ideally, we would want the average time taken to recover a loss to be around one round trip time (RTT). If that property holds, the network is not congested in the sense that loss recovery is functioning properly. If it takes more than few RTTs to recover the losses, then the network is more likely to be congested. Since the expected number of packets received during one RTT is $r_i RTT_i$, if the sum of out-of-order packet list length and the missing packet list length is $r_i RTT_i$, then roughly one RTT has passed since the first unrecovered loss (recall that r_i is the rate assigned to source i; we have omitted the time dimension in the notation for simplicity). Denote by F_i the length of source i's out-of-order packet list, and by G_i the length of source i's missing packet list, at some instant.

Then,

$$V_i = \frac{F_i + (G_i - 1)}{r_i RTT_i}$$

is a measure of the number of RTTs it would take to recover the loss. RCRT uses the exponentially weighted moving average of this value V_i as the measure of average congestion level, denoted by C_i, for source i. Thus, $C_i = 2$ means that it takes around 2 RTT_i's on average to repair losses for node i. (Section 3.7 explains how RTT is estimated.)

RCRT detects congestion by using a simple thresholding technique. If the C_i exceeds an upper threshold U for *any* source i, we say that the network is congested. RCRT declares the network under-utilized when C_i falls below a lower threshold L for *every* i. The gap between U and L is the desired congestion level in steady state that allows for some losses to achieve higher throughput while ensuring timely loss recovery. RCRT updates the C_is for every flow whenever a packet is received from that flow. It then decides whether the network is congested or under-utilized or otherwise, based on the algorithm described above. RCRT performs different actions in the congested and under-utilized states, as described in Section 3.5.

We use $L = 0.2$ and $U = 2$ as our thresholds, and the intuition behind this is as follows. The lower threshold $L = 0.2$ corresponds to the case where a lost packet takes around one RTT to recover and roughly one packet gets lost every five successful data packets ($\frac{(1+0+0+0+0)}{5} = 0.2$). If the packet losses are less frequent than this and are recovered well, the network should not be congested. The upper threshold $U = 2$ corresponds to the case where a packet is lost every five successful data packets, and this loss is recovered only after around four RTTs ($\frac{(1+2+3+4+0)}{5} = 2$). On the one hand, thresholds should be conservatively large since a flow's RTT_i increases dramatically when the network transitions from an uncongested to a congested state. On the other, small thresholds can account for the fact that C_i is lower than the actual time to recover loss since it is averaged over successfully received packets as well as the losses. Thus, large values for L and U will increase the rates faster, but may result in oscillatory behavior. Smaller L and U will allow RCRT to be more stable, but may require a longer convergence time. A high L and low U may be fast and stable, but the smaller gap between them will lead to frequent rate changes that will incur higher feedback overhead. We experimentally choose our threshold values to balance these constraints and ensure that RCRT reacts to congestion in a timely manner, but does not react to congestion earlier than it should.

3.5 Rate Adaptation

The second major challenge in RCRT is the design of a mechanism to adapt transmission rates in response to congestion (or lack thereof). Rate adaptation techniques have been studied widely in the literature. Most transport protocols additively increase flow rates (or, as in TCP [USC/ISI 1981], windows) in the absence of congestion, and multiplicatively decrease flow rates when congestion is detected. Chiu *et al.* [Chiu and Jain 1989] show that this AIMD approach guarantees stability and convergence to a fair and efficient allocation. While RCRT uses AIMD, it adapts the total aggregate rate of all the flows as observed by the sink, rather than

the rate of a single flow. In spirit, this is similar to XCP [Katabi et al. 2002], but there are some qualitative differences: XCP's decisions are made at a bottlenecked router, and XCP adapts rate based on the excess capacity at a link. We describe RCRT's rate adaptation design now.

Let $R(t)$ denote the sum of the currently assigned rates $r_i(t)$ for all flows i. RCRT uses AIMD on $R(t)$. When the network is not congested, RCRT increases $R(t)$ additively,

$$\text{Increase:} \quad R(t+1) = R(t) + A$$

where A is a positive constant[2], independent of the number of flows in the network. When the network is congested, RCRT decreases $R(t)$ multiplicatively:

$$\text{Decrease:} \quad R(t+1) = M(t)R(t)$$

where $M(t) \in (0, 1]$ is a time-dependent multiplicative decrease factor. The symbol t represents the discrete time instants at which RCRT makes rate adaptation decisions. An alternative design would have been to individually control the rates of each flow. However, since the sink has a greater perspective into network performance, designing the rate adaptation to control the total aggregate rate of the network allows the control algorithm to be independent of the number of flows, and less oscillatory when there are a large number of flows.

Two questions remain: When are rate adaptation decisions made? How is $M(t)$ determined? We address these questions next.

Whenever RCRT determines the network is congested (Section 3.4), it applies the rate decrease step described above, computes a new rate allocation for all the flows (Section 3.6), and sends the new rate r_i for each flow to the corresponding source. It doesn't make another rate reduction decision until it observes the effects of the previous decision. To do this, RCRT waits conservatively for at least three times the maximum value of RTT_i. This usually allows enough time for the rate feedback to reach the sources, and for the sources to send enough packets so that congestion measures at the sink are appropriately updated. Thereafter, if a packet is received from some flow i, and C_i is still above the upper threshold U, another rate decrease step is triggered, but only if f_i reports that it is using the rate assigned in the previous rate adaptation step. This last check ensures that RCRT does not react to stale information; especially in times of congestion, RCRT's RTT estimates can be slightly off and this step ensures that RCRT reacts only after the source has had a chance to respond to the previous rate adaptation. Finally, if the network is under-utilized, RCRT applies the increase rule above, but only if three times the maximum RTT_i plus three times the current inter-packet interval ($1/r_i$) has expired since the last rate adaptation decision. In the rate regime at which sensor networks operate, the inter-packet interval can sometimes be higher than the RTT; using three inter-packet intervals reduces the frequency of feedback generation during rate increase, at the expense of a more conservative increase. RCRT also refrains from increasing the rate if there are un-recovered burst losses [3] in any single node, even if their C_i is below L. We added this rule because burst losses at low C_i are

[2]We have used A = 0.5 pkts/sec for our experiments.

[3]We currently define three consecutive losses as burst losses.

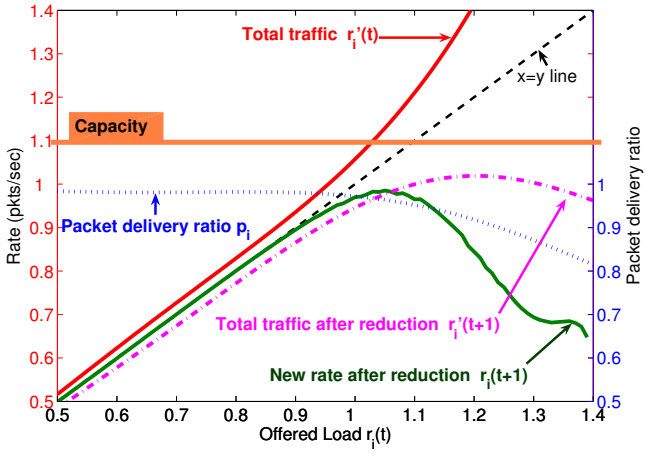

Fig. 2. Plot of p_i, corresponding total traffic $r_i(t)' = \frac{r_i(t)(2-p_i)}{p_i}$, and effect of $M(t)$ where p_i has been polynomial curve-fitted from the experimentally collected data. $r_i(t+1)$ is the new rate after $M(t)$ reduction, and $r_i'(t+1)$ is the expected total traffic after $M(t)$ reduction. X-axis is the offered load r_i. Y-axis represent both packet delivery ratio p_i (right axis), and the expected total traffic $r_i(t)'$ (left axis).

often a symptom of the onset of congestion, when a flow moves from uncongested state to a congested state.

In most transport protocols, the multiplicative decrease factor is constant (0.5 in TCP). However, because RCRT has a comprehensive view of network behavior across different sources, it can do better than simply halve $R(t)$. In RCRT, when a packet is received from flow f_i and that packet causes C_i to exceed the upper threshold U, $M(t)$ is computed based on the loss rate experienced by f_i. What is the intuition for this? Suppose that f_i's packet delivery ratio is p_i. This means that, every second, $r_i p_i$ packets out of r_i are being delivered to the sink without end-to-end loss. Furthermore, feedback traffic, roughly proportional to $r_i(1-p_i)$ must be delivered by the sink to the source in response to these losses.[4] Then the expected amount of traffic to and from the sink is $\frac{r_i}{p_i}$ and $\frac{r_i(1-p_i)}{p_i}$ respectively. This adds up to total traffic of at least $r_i' = \frac{r_i(2-p_i)}{p_i}$ for node i in the network.

We know that this level of traffic $(r_i'(t))$ was not sustainable, since the flow was congested. But, the last time that a rate adaptation decision was made, a source rate of $r_i(t)$ was deemed sustainable, assuming no end-to-end losses. Given a packet delivery rate p_i, it would now be safe to set f_i's rate such that the *total amount of traffic* is $r_i(t)$. To do this, we should reduce f_i's rate to: $r_i(t+1) = \frac{p_i}{2-p_i} r_i(t)$, so that $r_i'(t+1) \equiv r_i(t)$. However, RCRT is a little bit more conservative than that. It reduces the *overall* total rate $R(t)$ by that factor, by setting the multiplicative

[4]In RCRT, a list of missing information can be packed into a single NACK. In this analysis, we assume that only one missing packet is included in each NACK packet. This can happen when the losses are infrequent and spread out uniformly over time.

decrease factor $M(t)$ to be:

$$M(t) = \frac{p_i(t)}{2 - p_i(t)}$$

where f_i is the flow that triggers the rate decrease. $M(t)$ is larger than 0.5 for all p_i greater than 0.67. So, in regimes where the end-to-end packet reception rate is high, RCRT can more efficiently adapt to congestion than protocols that employ a fixed multiplicative decrease factor of 0.5.

We now give some intuition for how $M(t)$ behaves for different values of $r_i(t)$. To do this, we conducted an experiment consisting of 40 nodes, and measured the *average* or smoothed value of p_i as a function of r_i. This is shown in Figure 2. Also shown in that figure is 1) the *total* network traffic corresponding to a given r_i (labeled $r'_i(t)$), 2) the value $r_i(t+1)$, which is the rate that would have been assigned to node i by RCRT after its multiplicative decrease, and 3) the value $r'_i(t+1)$, which is the expected total network traffic corresponding to $r_i(t+1)$ after the multiplicative decrease. For this network, the network was empirically-determined to be saturated at 1.1 pkts/sec per node (Section 4). Note that the estimated total traffic $r'_i(t)$ increases drastically as the offered load r_i increases above this capacity line. We'd like to point out two important properties of $M(t)$. First, regardless of the value of r_i, $M(t)$ is always successful in assigning a rate $r_i(t+1)$ which keeps the aggregate rate below overall capacity, but also ensures that the expected total traffic $r'_i(t+1)$ is below capacity. Second, $M(t)$ is more aggressive when $r'_i(t)$ is well above the network's capacity. These two properties imply that $M(t)$ is sufficient to move RCRT out from a congested state to an uncongested state regardless of when the congestion was detected and rate decrease was triggered. Moreover, this detection is rapid: even if several packets are lost due to congestion, a single successful delivery suffices to trigger a rate decrease.

Loss Rate Estimation. Thus far, we have not described how $p_i(t)$ is calculated. RCRT keeps track of estimated path loss rate $(1 - p_i(t))$ for each flow using the *Average Loss Interval* method discussed in [Floyd et al. 2000]. It uses the average interval (in number of packets) between loss events to estimate the loss rate of a flow. Denote the interval between m-th and $m+1$-th loss on flow i as $s_{i,m}$. Then, for the recent $1 \leq m \leq n$ losses, the average loss event interval $\hat{s}_i$ for flow i is calculated as,

$$\hat{s}_{i(1,n)} = \frac{\sum_{m=1}^{n} w_m s_{i,m}}{\sum_{m=1}^{n} w_m}$$

$$\hat{s}_{i(0,n-1)} = \frac{\sum_{m=0}^{n-1} w_m s_{i,m}}{\sum_{m=1}^{n} w_m}$$

$$\hat{s}_i = max(\hat{s}_{i(1,n)}, \hat{s}_{i(0,n-1)})$$

where s_0 is the interval since the most recent loss and w_m is the weight given to each loss interval in the history. The larger of the two quantities $\hat{s}_{i(1,n)}$ and $\hat{s}_{i(0,n-1)}$ is selected so that the most recent loss with small s_0 does not cause severe underestimation of $\hat{s}_i$. We have followed the analysis in [Floyd et al. 2000] and used $n = 8$ and $w = [1, 1, 1, 1, 0.8, 0.6, 0.4, 0.2]$ as our parameters. Intuitively, these

choices give greater weight to recent loss periods, and lesser weight to more distant loss events. Then we compute $p_i(t)$ from

$$1 - p_i(t) = \frac{1}{\hat{s}_i(t)}$$

We chose the Average Loss Interval (ALI) method over others after experimentally verifying that it was more robust to parameter choices than the alternatives. A window-based method that estimates loss rates based on some window of the number of packets or time is highly sensitive to the choice of window. An EWMA approach is sensitive to the choice of gain, and includes a significant history of losses. By contrast, the ALI method always looks at a fixed number of loss events, and preferentially weighs the recent ones. This helps RCRT be more responsive to the onset of congestion.

3.6 Rate Allocation

Once the total rate $R(t)$ is calculated by the rate adaptation mechanism, the role of RCRT's rate allocation component is to implement the capacity allocation policy P_j associated with its sink. This is a novel aspect of RCRT; to our knowledge, most prior work has (implicitly or explicitly) embedded a capacity allocation policy within the rate allocation mechanism.

RCRT's rate allocation component essentially assigns rates $r_i(t)$ to each flow in keeping with the rate allocation policy P_j such that the individual flow rates sum up to $R(t)$. Because RCRT decouples rate adaptation from rate allocation, it is possible to obtain this flexibility. We see this flexibility as being crucial, since it is unclear that there is, *a priori* a preferred policy for sensor networks (unlike the Internet, which is a shared infrastructure, and for which some form of fairness makes sense). RCRT's rate allocation component is similar to XCP's [Katabi et al. 2002] fairness controller, but offers greater flexibility.

Our current prototype contains three different policies.

Demand-proportional. In this policy, each flow expresses a desired rate, that we call its *demand* d_i. This policy allocates rate r_i to each node i proportional to its demand d_i such that the fraction $(r_i(t)/d_i)$ is the same for all i;

$$r_i(t)/d_i = \rho(t) \quad \forall i \in \mathbb{S}$$

As long as we use same $\rho(t) = \rho_i(t)$ for all node i, demand-proportional allocation follows. We compute $\rho(t)$ as follows:

$$\begin{aligned} D &= \sum_{i \in \mathbb{S}} d_i \\ R(t) &= \sum_{i \in \mathbb{S}} r_i(t) \\ \rho(t) &= \frac{R(t)}{D} \end{aligned} \tag{1}$$

where D is the sum of all d_is. Then, the new rate for each node i simply becomes:

$$r_i(t) = \rho(t) \cdot d_i$$

Demand-limited. In this policy, $R(t)$ is divided among all the flows equally, except that no flow gets a higher rate than d_i. Given $R(t)$, there exists a simple greedy algorithm for computing a demand-limited rate allocation.

Fair. This allocation policy assigns an equal share of $R(t)$ to all flows, regardless of d_i. Flow demands are ignored in this policy.

As an example, consider a scenario where two nodes A and B demands for 1 pkt/sec and 2 pkts/sec respectively, and the network can only support 2.4 pkts/sec total. Then, the *Demand-proportional* policy will allocate 0.8 and 1.6 pkts/sec, the *Demand-limited* policy will allocate 1.0 and 1.4 pkts/sec, and and the *Fair* policy will allocate 1.2 and 1.2 pkts/sec, to nodes A and B respectively.

While we have not experimented with other policies, our prototype software is written modularly to easily accommodate other policies. For example, it is easy to implement a weighted allocation policy, where each source gets a rate allocation proportional to a configured weight w_i (weighted allocation is similar to demand-proportional allocation, with the only difference that in the latter, a source is not assigned a rate higher than its demand). This weighted allocation policy is a generalized form of priority allocation, in which some nodes are given higher priority than others. Finally, new rate allocation policies should not require any changes to the protocol (or to code on the sensors) — they can simply be configured at the sink.

3.7 Other Details

In our description, we have left out several details of RCRT. We describe them here briefly for completeness.

Source Node Behavior: In RCRT, each source node initiates a flow by first establishing an end-to-end connection with the sink. RCRT uses a three-way handshake connection establishment mechanism similar to that of TCP where the third ACK is substituted with the first data packet. While doing this, the source tells the sink its desired data rate d_i (although this information can, in principle, also be just configured at the sink), and the sink tells the node the rate r_i. This three-way handshake also allows RCRT to guess an initial RTT estimate. Once a connection has been established, the node transmits packets on this connection. Each node originates data at rate at most r_i assigned by the sink. The rate r_i is enforced by a token bucket with rate r_i. Each packet contains sequence number, flow ID, and the rate of the corresponding flow. The rate r_i regulates only the new packets that are sourced by this node. End-to-end retransmissions are not constrained by this rate, nor are forwarding and link-level retransmissions performed the lower layers of the protocol stack.

Since rate allocation and loss recovery are controlled end-to-end by the sink, each node simply reacts to the sink. When a node receives a packet from the sink with *new rate* r_i' (this is usually sent in a feedback packet which may or may not contain NACK information, see below), the node adjusts the token bucket rate accordingly. When a node receives a feedback packet with *NACK* information, the

node retransmits those missing packets.

Finally, each source has a fixed-size retransmit buffer which contains sent packets that have not been acknowledged. Clearly, a source cannot store an infinite number of packets for potential retransmission. To efficiently manage the retransmit buffer, each feedback packet (described below) includes a cumulative ACK sequence number, which indicates the last sequence number that the sink has received contiguously without any missing packets. The source can safely remove packets in the retransmit buffer that are covered by the cumulative ACK sequence number.

When the retransmit buffer is full, the source stops sending new packets and retransmits the packet whose sequence number is one higher than the last acknowledged packet at a rate of $min(\frac{r_i}{2}, \frac{1}{RTT})$. The source does this until it is told that the sink has received those packets and hence it is safe to overwrite the buffers. When this packet has been received at the sink, loss recovery is triggered, the rate is adjusted for all nodes, and a feedback packet is sent. The retransmit buffer rarely fills up since a cumulative ACK is piggybacked in every feedback packet. Additionally, to prevent a stall, each source requests explicit feedback from the sink when its retransmit buffer is more than half full. It does this by setting a bit in every outgoing packet. To minimize the number of feedback transmissions while allowing for enough packets to be in transit, the buffer must be large enough ($>> 2r_i/RTT_i$ packets).

Feedback packets: Every feedback packet sent to a source i contains the assigned rate r_i, a list of $NACK$ed sequence numbers, a cumulative ACK, and the RTT_{avg} (see below) value for that node. Thus, every feedback packet is completely self-contained so that, even if one of these is lost, a subsequent feedback packet suffices to adjust the node's rate, and recover from loss.

RCRT has to be careful in sending feedback packets, since they represent significant overhead. When a packet is received at the sink from node i, the sink sends a feedback to the node only when at least one of these four conditions have been met: one or more missing packets have been detected; the node is sending at a rate different from the assigned rate; a duplicate packet with an already acknowledged sequence number has been received; or, a feedback packet has been explicitly requested by the node. However, RCRT is careful not to send feedback more often than once every RTO_i, defined as $RTT_{avg} + 2 * RTT_{var}$, where RTT_{avg} is the average RTT (see below), and RTT_{var} is the mean deviation of RTT. This rate limit trades-off increased convergence time for lower overhead, especially in times of congestion. Finally, recall that rate adaptation decisions are made on RTT time-scales, also reducing the rate of sending feedback packets.

RTT estimation: Finally, the sink estimates the RTT of each node using feedback packets. RCRT does not require any time synchronization mechanism to do this. The sink records the time T_i when it sends a feedback packet to node i. Node i remembers the time T_i' at which it had received the feedback. When node i next sends a packet to the sink at time T_i'', it calculates the interval $T_i'' - T_i'$ and piggybacks this value in the packet. Upon reception of this packet at time T_i''', the sink can calculate the instantaneous RTT sample value $RTT_{inst,i}$ by $(T_i''' - T_i) - (T_i'' - T_i')$. RCRT uses an exponentially-weighted moving average of this value to get the estimated $RTT_{avg,i}$ for node i. Earlier in this section, when we have referred

to RTT_i, we have meant $RTT_{avg,i}$.

3.8 Discussion

Three minor details of RCRT are worth mentioning. One is that any NACK-based scheme suffers from not being able to detect the loss of the last packet (or the contiguous loss of the last few packets), since it relies on the successful reception of a later packet. In RCRT, we detect and recover from such losses during connection tear-down: after they are done sending data, sources explicitly tear-down the transport connection, at which point they synchronize sequence numbers and repair the last loss (if any). The second is that in large networks, the maximum RTT can be high and since RCRT makes decisions on RTT timescales, the network may converge slowly. The third is the case when several consecutive packets are lost due to high congestion. Until a subsequent data packet is received, the sink will not notice the congestion, hence RCRT's reaction may be delayed. RCRT's reaction may also be delayed when several consecutive feedbacks are lost due to high congestion. However, at least one packet from at least one of the congested nodes will eventually be delivered, and this will bring the rates down sufficiently ($M(t)$, Section 3.5). The reduced rates will be applied to at least the nodes that can receive the feedback packets, and this in turn will make some capacity room for the feedback packets to reach the other congested nodes. Hence, RCRT will re-gain control.

A fourth detail requires a little more explanation. Our current RCRT implementation uses link-level retransmissions, and a natural question to ask is: How would RCRT perform in the absence of link-layer retransmissions? To us, this question is ill-posed: in wireless networks with link loss rates approaching 30%, trying to design an end-to-end ARQ-based reliability mechanism without link-layer retransmissions is a fundamentally bad idea, since it would rely on expensive end-to-end retransmissions to repair loss. That said, with a limit on the number of retransmissions as in RCRT, there may be scenarios in which the end-to-end packet delivery rate is still low. In this case, p_i (Section 3.5) will be low, and the multiplicative decrease factor could be lower than 0.5 (the constant multiplicative decrease used by TCP). Thus, in this regime, RCRT may be more conservative than TCP's AIMD, but that mechanism it not known to work well in lossy conditions anyway; it is unrealistic to expect efficient *and* reliable delivery in a network where the packet delivery ratio is extremely poor. In those conditions, the best approach would be to re-provision the network by re-deploying some nodes, or adding capacity using extra nodes or introducing tiers. We evaluate the effect of link-level retransmissions in Section 4.4.

Finally, we address the question: does RCRT really avoid congestion collapse? There are two cases to consider. When source nodes hear feedback from the sink, RCRT's multiplicative decrease function $M(t)$ (Figure 2) aggressively reduces source sending rates, allowing the network to recover from congestion. When a source does not hear feedback, the source sends at $r_i(t)$ for a while, but it eventually fills up its retransmit buffer and stalls, effectively sending one packet per RTT (Section 3.7). This is a conservative solution, since the source needs to "probe" by sending at least one packet to recover from transient path failures, and the RTT is the right timescale to do this. (Of course, more conservative approaches, like

exponentially backing-off the probe interval are possible, and we have left these to future work). When it is stalled, the source does not congest the network, allowing the network to recover (if, indeed, the loss of feedback packets was caused by congestion).

There are several open questions in the design of RCRT. First, we have not examined inter-sink cooperation. Having such a mechanism in place would help administrators control capacity allocation across different applications and provide higher efficiency gains. Second, our current design does not support a policy which gives excess bandwidth to unconstrained sources. In most of the topologies we have experimented with, almost all sources are constrained by one bottleneck wireless region. However, there can exist topologies where some sources may not be so constrained, and it might be beneficial to allocate higher rates to these sources while rate limiting the congested sources only. Since RCRT does not have insight into the network, it cannot easily distinguish between these sources. Even if the sink observed a set of uncongested flows, it cannot determine how those flows interfere with other flows. So, trying to allocate higher rate to those flows might exacerbate congestion on other flows.

Finally, we conclude with a discussion of an important aspect of congestion control design that we have ignored so far: what should the "application" do if the data generation rate exceeds the rate offered by the congestion control protocol? Or, how should the application adapt to the rate control performed by the network stack? Prior work in congestion control for WSN has assumed infinite backlog of data that can be arbitrarily delayed or dropped. However, we argue that the application should adapt to rate control. Applications can adopt many methods to ensure that the offered rate matches the rate that can be supported by the underlying network. Simple periodic sensing applications can sub-sample the stream so that the user-perceived quality degrades more gracefully. However, such subsampling should be accompanied by appropriate low pass filtering to avoid altering the spectral characteristics of the data. Bulk transfer applications can either adjust the chunk generation rate, use adaptive compression techniques, or employ local processing. To support such adaptation, a transport protocol must provide precise and timely information about the currently achievable rate to the application, and RCRT is capable of providing this information.

4. EVALUATION

In this section, we present results from an extensive performance evaluation of our implementation of RCRT on a wireless sensor testbed.

4.1 Implementation and Methodology

We have implemented RCRT in TinyOS for the motes[5] and in C for a PC-class sink device running Linux. The RCRT module on the motes provides a transport-layer interface that a sensor application can use to initiate a flow to the sink and send data packets. Also, the module implements a token bucket, whose rate is set to the rate allocated by the sink, to regulate data packets generated at that node. The

[5]RCRT has been implemented in both TinyOS 1.x and TinyOS 2.x and their performance are similar.

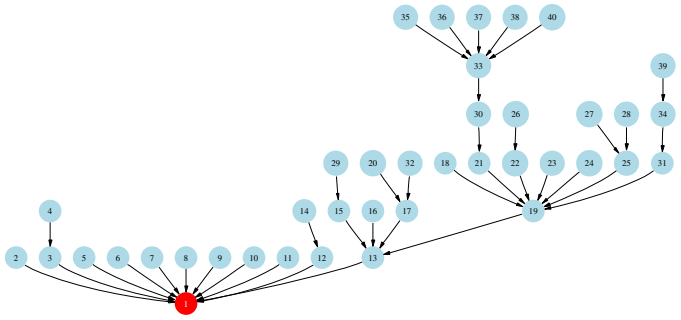

Fig. 3. A snapshot of routing topology constructed by MultihopLQI

memory footprint of RCRT for the mote is approximately 5252 code bytes and 374 bytes of RAM for a packet payload size of 64 bytes. The code size excludes RCRT-independent basic components such as timer, flash, MAC, and routing. It uses 64KB of external flash for a retransmit buffer. All other mechanisms described in Section 3 including loss detection, rate adaptation, rate allocation, congestion detection, and RTT estimation are implemented at the sink.

We have evaluated our RCRT implementation on an indoor wireless sensor network testbed[6]. Most experiments are conducted on a 40-node subset of this testbed, but we also show results from some larger topologies. Each sensor node in our testbed is a Moteiv Tmote with an 8MHz TI-MSP430 micro controller, IEEE 802.15.4-compatible CC2420 radio chip, 10KB RAM, and a 1MB external serial flash memory. These motes are deployed over 1125 square meters of a large office floor.

We have used MultihopLQI in TinyOS as our routing tree protocol and default CSMA in TinyOS as our MAC protocol for our experiments. In MultihopLQI, each node dynamically selects its parent to construct a routing tree to the base station using the link quality indicator provided by the CC2420 radio chip. Since a sink-to-mote reverse path is required in RCRT (for the feedback packets), we have added a data-driven reverse path construction mechanism. This works as follows. Each node maintains a routing table. When it receives a packet with source address S from a neighbor N, it adds a route entry to S with next hop N. Feedback packets are forwarded using this routing table. Finally, our implementation uses link-layer retransmissions based on chip-level acknowledgments with up to 4 retransmissions unless stated otherwise. We experiment with different link-layer retransmission strategies in Section 4.4.

Figure 3 is a snapshot of the routing tree constructed by the MultihopLQI during one of our experiments. Due to changes in wireless link quality over time, the routing tree changes. Figure 14 shows how frequently each node changed its parent in one of our experiments; our experiments were conducted in a dynamic environment, with significant routing variability. However, in most of our experiments, the routing protocol consistently produced 5 to 7-hop deep routing trees. We show the effect of different topologies/node density in Section 4.3.

[6]Tutornet: Tiered Wireless Sensor Network Testbed. (http://enl.usc.edu/projects/tutornet/)

Design Goal	Section
Reliable end-to-end transmission	All
Network Efficiency	Section 4.2.1 Section 4.2.2 Section 4.2.5 Section 4.3
Support for concurrent applications	Section 4.2.4
Flexibility	Section 4.2.4 Section 4.2.3
Robustness	Section 4.2.3 Section 4.3

Table III. Experiments to demonstrate that RCRT meets its goals.

In each of our RCRT experiments, each source originated at least 1000 data packets. This traffic is synthetic, and does not represent the workload generated by any sensor; however, since RCRT is oblivious to the actual data, this is an appropriate methodology. Each experiment ran from 30 minutes to an hour depending on the achieved rate. We logged every packet received at the sink along with the current allocated rate at the time of packet reception.

4.2 Results

In this section, we describe experimental results that validate our RCRT design, demonstrate that RCRT achieves the goals discussed in Section 3.1, and show that RCRT outperforms the state-of-the-art in distributed congestion control. Table III summarizes our methodology: for each high-level goal described in Section 3.1, we have designed at least one experiment to validate or quantify that goal. (One exception is the goal of minimal sensor functionality, which follows from RCRT's design). Some experiments are used to validate or quantify more than one goal.

In most cases, we evaluate RCRT's performance using its *rate allocation profile* which is the plot of the assigned sensor rates r_is as a function of time. In some cases, particularly to show the efficacy of RCRT's capacity allocation policies, we plot the average goodput achieved by the node during the experiment.

We must emphasize that we have run RCRT experiments under very general settings. All experiments reported here are from an actual implementation running on a real testbed. The underlying routing and MAC layers are not optimized in any way, nor have the experiments been run at special times to avoid interference. Our testbed is susceptible to significant interference both from other 802.15.4 radios and from 802.11 radios, and this interference is highly time-varying.

Finally, we note that RCRT achieves 100% reliable packet delivery *in all experiments* we have conducted for this paper. For this reason, we do not focus on RCRT's end-to-end reliable transmission mechanism, but instead focus on how well RCRT's congestion control works: what rates are assigned, how RCRT reacts to dynamics, and so on.

4.2.1 *Baseline.* We start with a simple baseline experiment that illustrates some of the important features of RCRT. In this experiment, RCRT runs on a 40-node network. One node is a base-station, and the others are programmed to send data

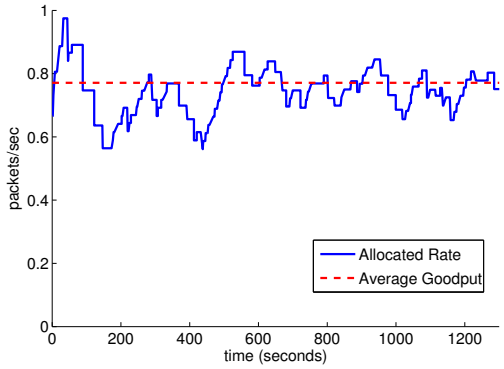

Fig. 4. Rate r_i allocated to every node in the 40-node experiment with fair rate policy

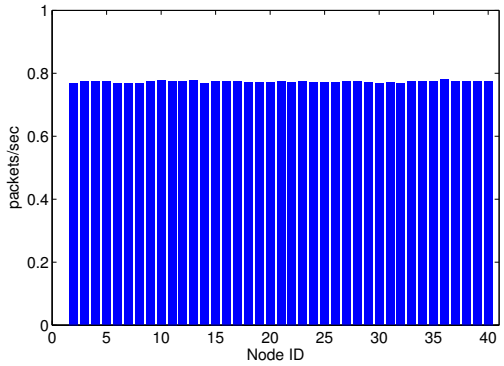

Fig. 5. Per-node goodput in the 40-node experiment with fair rate policy

back to the sink. The fair rate allocation policy is used at the sink.

Figure 4 shows the rate allocation profile $r_i(t)$ allocated to every node by the sink as a function of time. The solid line represents the instantaneous allocated rate logged at the time of every packet reception. The dashed line shows the average achieved goodput from all nodes. Since a fair-rate policy was used, all nodes received the same goodput. This graph shows the efficiency of RCRT's rate adaptation mechanism. Unlike other AIMD schemes that drastically reduce the rate in response to congestion by halving the rate (or, as in TCP, the window) RCRT tries to stay near the steady state average rate by making small adaptive reductions. At the beginning of the experiments, the allocated rate over-shoots to almost 1 pkts/sec, and drops down to around 0.55 pkts/sec (almost half). This is because when the nodes first start sending packets, the network was not congested, and the RTT estimate takes some time to stabilize. But after this transient, the allocated rate converges and stays within 25% of the average goodput. This less oscillatory behavior results from RCRT's rate adaptation design which makes rate adaptation decisions based on the overall traffic, rather than on a single flow.

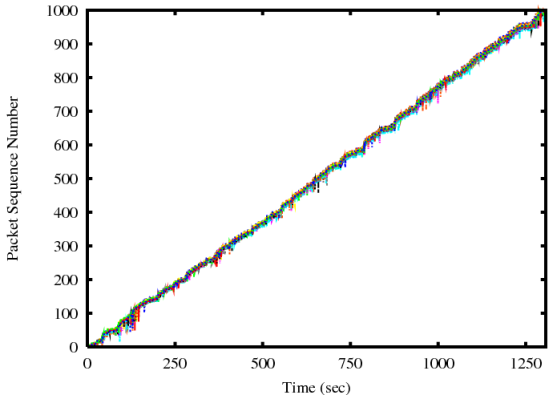

Fig. 6. Packet reception plot for all nodes in the 40-node experiment with fair rate policy

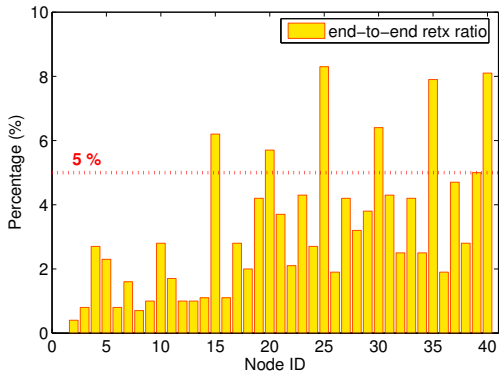

Fig. 7. Percentage of packet repaired by end-to-end loss recovery mechanism in the 40-node experiment

Figure 5 shows the per-flow goodput achieved at the sink. Each bar represent the average goodput achieved by each source during the entire experiment. This graph shows that nodes achieved approximately fair goodput: the difference between the largest and the smallest goodput is only 0.015pkts/sec! This is a surprising result since one might expect that allocating *same* sending rate to all nodes in a multi-hop environment would penalize nodes farther away from the sink, since they traverse more hops. RCRT maintains fairness because its rate adaptation mechanism conservatively adapts to the source that experiences congestion most along the path to the sink.

Figure 6 shows the packet reception plot for all the nodes in the network. Each point on the curve is the time at which a packet with a particular sequence number was received. Since all of the packets were eventually delivered, the progress in sequence numbers approximately corresponds to the progress in number of packets received. The slope of each plot is the estimate of the instantaneous goodput that each node achieves at that point in time, and this figure shows that all nodes have

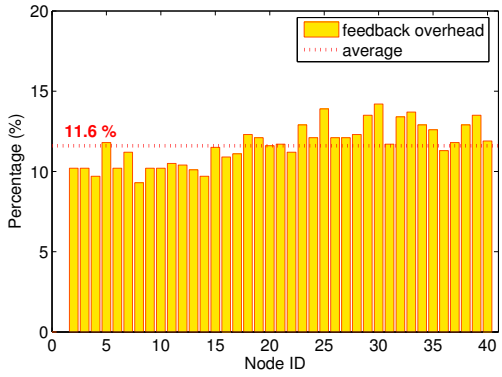

Fig. 8. Feedback packet overhead (percentage relative to data packets) in the 40-node experiment

approximately fair goodput throughout the experiment. The small spikes below the curve represent lost packets being repaired by RCRT's end-to-end loss recovery mechanism.

How many packets are recovered via end-to-end retransmission? Because of link-layer retransmissions, for all but 6 nodes, only 5% of the packets incurred end-to-end retransmissions, and for none of the nodes were more than 9% of the packets recovered end-to-end (Figure 7). This is encouraging, since one of the original design goals of RCRT's end-to-end reliability mechanism was to avoid a feedback implosion. We can quantify the overhead of feedback in RCRT. In this experiment, 4549 feedback packets were sent *in total* for 39000 data packets, representing an overhead of 11.6% (Figure 8. This feedback was used *both* to recover from losses and to adapt to congestion. Contrast this with TCP, in which every connection can see *half* as many ACK packets as data packets (most TCP implementations acknowledge every other packet).

4.2.2 *Optimality.* The baseline experiment demonstrates some of the salient features of RCRT's algorithms. The next question we address is: How close does RCRT's rate allocation get to the optimal? One way to evaluate the performance achieved by RCRT is to determine the maximum fair and reliable rate sustainable on the same network with same routing and MAC layers. We address this question by experimentally evaluating the goodput received by two different kinds of transport mechanisms at different offered loads. *Best-effort* transport sends data at a configured rate, but includes no end-to-end reliability and does not adapt to congestion. *Reliable* transport sends data at a configured rate, includes end-to-end reliability, but does not adapt to congestion. In our implementation for the experiments, best-effort transport is essentially MultihopLQI with application-level sequence numbers, and reliable transport adds end-to-end loss recovery (Sec.3.3) on top of that.

Figures 9 and 10 plot the average goodput over all nodes for two transport mechanisms at various offered loads. The thick solid curve shows the average goodput achieved at the sink, the error-bars parallel to the y-axis indicate the maximum variation in node goodput at each offered load, and the straight dotted diagonal

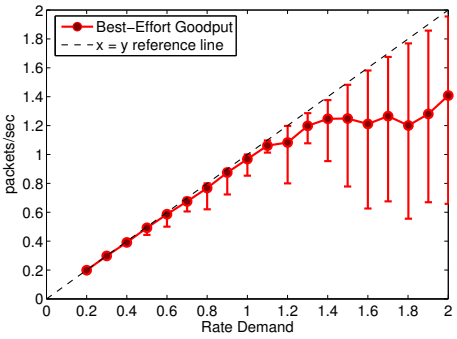

Fig. 9. Average goodput achieved by best-effort transport. X-axis is the rate at which each node sourced packets. Y-Error bar represent the maximum and minimum goodput among all nodes

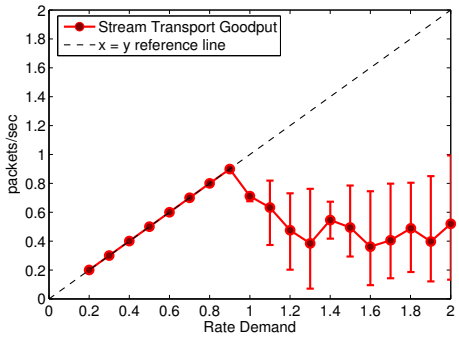

Fig. 10. Average goodput achieved by reliable transport. X-axis is the rate at which each node sourced packets. Y-Error bar reprint the maximum and minimum goodput among all nodes

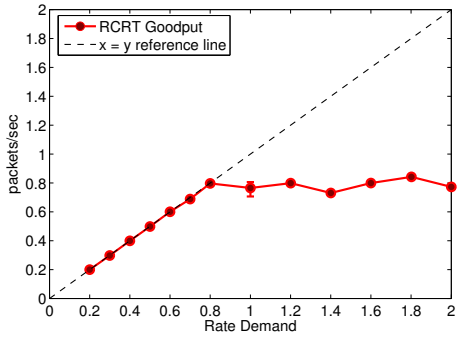

Fig. 11. Average goodput achieved by RCRT. X-axis is the demand assigned to each node. Y-Error bar represent the maximum and minimum goodput among all nodes

line represents the rate achievable with infinite capacity.

Figure 9 shows the maximum fair rate achievable without end-to-end reliability on our 40-node testbed. Until the offered load reaches 1.1 pkts/sec per node, all nodes achieve approximately fair goodput (small error bars) and 95.4% reliability (not shown). However, as the offered load increases above 1.2 pkts/sec, variability in goodput increases, and the reliability drops below 90%.

Figure 10 represents the achievable rate with end-to-end reliability, but no congestion control. Our network is able to sustain up to 0.9 pkts/sec per node. Thereafter, it experiences congestion collapse: acknowledgments and retransmissions use up much of the network capacity, resulting in less goodput and lower fairness than best-effort transport.

Since RCRT provides end-to-end reliability, we should compare its achieved rate with that of Figure 10. If we define 0.9 pkts/sec as the maximum sustainable rate for reliable transport, then, as Figure 11 shows, RCRT achieves nearly 0.8 pkts/sec

per node, or 88% of the sustainable rate for reliable transport. In this experiment, we assigned all nodes equal demand. The figure plots this increasing demand on the x-axis. Of course, since RCRT is congestion-adaptive, sources only send at the assigned rates, not at their demands.

Another way to evaluate RCRT's optimality is to consider a single-source network. We conducted a single-source single-sink experiment and found that best-effort transport can achieve up to 95 pkts/sec, and RCRT can achieve up to 60 pkts/sec on average (As an aside, note that RCRT is capable of achieving high goodput (60 pkts/sec). In our previous experiments, the low per-node goodput (0.8 pkts/sec) is mainly a function of the topology, not our protocol.) We can relate these experimental results to those observed on our larger topology as follows. In Figure 3, an instantaneous snapshot of the routing tree during one of our experiments, we estimate that node 19 is the most congested node: it has 19 children, 3 siblings, 3 children of the siblings, and hence a parent (node 13) that has total of 26 children. Let's say every node sends 1 unit of data per 1 unit of time. Then node 19 receives 19 units of data and sends 20 units of data (including it's own data) per unit time. Also, node 19 contends with 6 units of data from its 3 siblings and their children to reach its parent. Finally, node 19 contends with 27 units of data that the parent (node 13) transmits to its parent (node 1). Hence, the channel capacity around $19 \rightarrow 13$ must be shared *at least* by total traffic of 72 units of data per unit time ($= 19 + 20 + 6 + 27$), even if we assume an ideal MAC. (We call this 72 the maximum contention factor of this routing topology.) This means that the optimal sustainable rate in this network is $60/72 = 0.83$ for RCRT and $95/72 = 1.32$ for best-effort transport. These numbers match what we observe above, and confirms that RCRT closely achieves what the topology allows; RCRT in a 40-node network achieves 96% of what it can achieve in a single node network. We discuss how resilient RCRT is to the effect of network topology in Section 4.3.

In summary, RCRT manages to assign near-optimal rates by having congestion control functionality at the sink. We have compared the various transport protocols with the same radio, MAC, and routing layers, to ensure that our results are not affected by differences in the underlying protocol layers. Our results show that it is possible to estimate and manage overall network capacity in a centralized manner, and achieve high efficiency.

4.2.3 *Robustness.* In this section, we conduct an experiment that demonstrates RCRT's robustness, and also validates its flexibility in capacity allocation. In this experiment, nodes join and leave dynamically, and different nodes are assigned different demands. The network is configured to use a demand-proportional allocation policy. We set up three sets of flows that request different demands. Specifically, in this experiment, 31 flows start at time 0. Sixteen of these (which we will call set A) demand 1.0 pkts/sec and the other 15 flows (set B) demand 0.5 pkts/sec. The remaining 8 flows (set C) join in after 500 seconds with a demand of 4 pkts/sec. All flows send the same total number of packets, but set C finishes earlier because of its higher demand.

Figure 12 shows the rate allocation profile for this experiment. Recall that RCRT allocates exactly the same rate to all flows having the same demand. During the first 500 seconds of the experiment, sets A and B were allocated roughly the rate

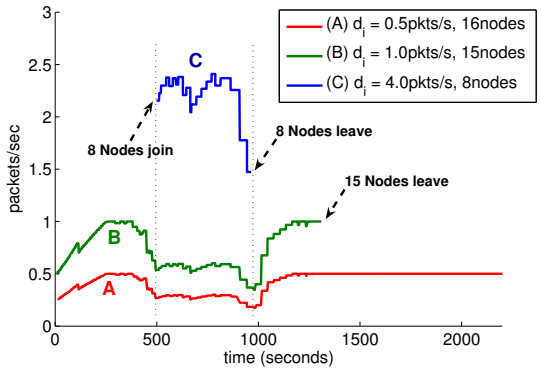

Fig. 12. Rate r_i allocated to each node in the 40-node experiment with demand-proportional rate allocation policy when 8 nodes join in after 500 seconds

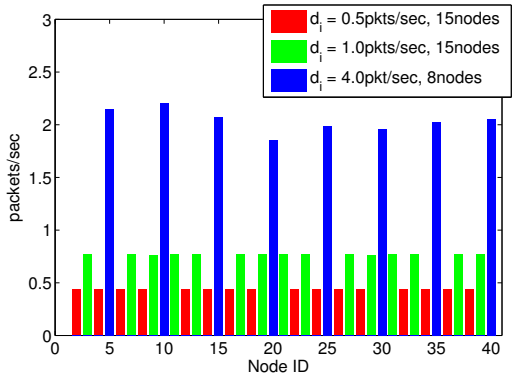

Fig. 13. Per-flow goodput in the 40-node experiment with demand-proportional rate allocation policy when 8 nodes join in after 500 seconds

that they demanded: 0.5 and 1 pkts/sec. When the third set of flows C joined in with significantly higher demand, the network immediately experienced congestion, and the flows in both A and B were forced to reduce their rates. Between 500 and 1000 seconds into the experiment, all three sets of flows were active, and were all allocated rates proportional to their demands. When the flows in set C had completed at around 1000 seconds, the network had enough capacity to satisfy the demands of flows A and B. This result shows that RCRT is robust to node joins and leaves, its congestion detection mechanism and the rate adaptation mechanism successfully adapted the network-wide aggregate rate to the network state, and the rate allocation mechanism indeed allocated rates proportional to the demands of each node.

Figure 13 plots the goodput achieved by each node for this experiment. While nodes with identical demands achieved comparable goodputs, the average goodput between different sets is, interestingly, *not* exactly proportional to their demands. Specifically, the average goodput achieved by sets A, B, and C is 0.44, 0.77, and

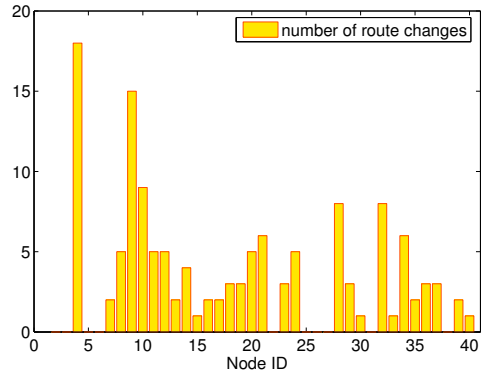

Fig. 14. Number of routing parent changes during the 40-node experiment with demand-proportional rate allocation policy when 8 nodes join in after 500 seconds

2.04 pkts/sec respectively, while their demands are 0.5, 1.0, and 4.0 pkts/sec respectively. This is because each set experienced network congestion for different fractions of their lifetimes. Since the flows in set C experienced congestion during their entire lifetime and only achieved half the goodput of their demand, the average $\frac{r_i}{d_i}$ during this congested period is about 0.5 ($2.04/4.0 \approx 0.5$). Since set B experienced congestion and was assigned half the demanded rate for half of its lifetime, its expected goodput is around 75% of what it would have achieved in an uncongested network. This roughly matches the goodput that set B actually achieved ($0.5\frac{1}{2} + 1.0\frac{1}{2} = 0.75 \approx 0.77$). Finally, set A was assigned half the rate for 1/4 of its lifetime, which amounts to $0.25\frac{1}{4} + 0.5\frac{3}{4} = 0.4325$ pkts/sec, which is close to the observed 0.44 pkts/sec. Thus, RCRT's demand-proportional allocation policy results in *instantaneous*, but not *long-term*, demand-proportional rate assignments. It is possible to implement a long-term demand-proportional allocation policy in RCRT, and we have left this to future work.

Finally, Figure 14 shows how frequently each node changed its routing parent during this 38-minute experiment. Even though, on average, each node changed its parent 3.4 times, RCRT assigned rates correctly to all the flows. This also highlights the robustness of RCRT's design and implementation.

4.2.4 *Flexibility.* In this section, we demonstrate that RCRT achieves two more of its original goals: that it can support multiple concurrent applications, and that each application can use different capacity allocation policies.

We ran two separate "applications" with two sinks. Each application ran on a different sink and used different rate allocation policies: one used demand-proportional allocation, and the other used demand-limited allocation. The two sinks at the upper-tier were connected via 802.11b wireless. Nodes 15 and 30 were the gateway motes on our testbed connected to the two sinks. Each sink was a Stargate running Linux. The motes used a multi-sink version of MultiHopLQI, so that two trees were formed, one rooted at each sink. We used additional routing software that allowed both sinks to receive data packets from *all* motes. Thus, this set up represents two applications running on a tiered network.

Set	App. ID	Demand	Num.pkts	Num.nodes
A_1	1	1.0 pkts/sec	1000	19
B_1	1	0.5 pkts/sec	500	18
A_2	2	1.0 pkts/sec	500	19
B_2	2	0.5 pkts/sec	250	18

Table IV. Setup for two-application, two-sink experiment

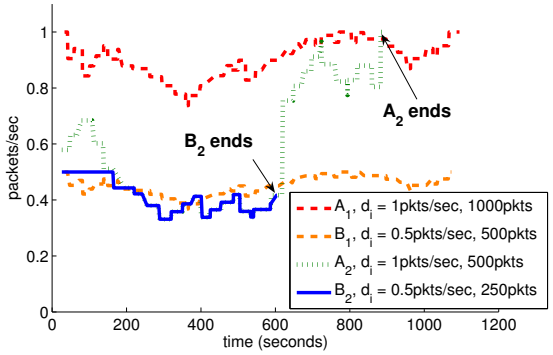

Fig. 15. Rate allocated to each node in 39-node experiment with two applications running on two sinks

In this experiment, we used 37 motes. The experiment was set up as shown in Table IV. The demand-proportional application comprised two sets of nodes A_1 and B_1, the first set being assigned twice the demand as the second. The demand-limited application comprised the same two sets (denoted A_2 and B_2) respectively, with identical respective demand assignments. The total aggregate demand was 56 pkts/sec, enough to saturate the network.

Figure 15 shows the rate allocation profile for this experiment. Flows in application 1 (sets A_1 and B_1) were assigned rates proportional to their demands, and flows in application 2 (sets A_2 and B_2) were assigned rates limited by their demands. Thus, notice that even though flows in A_1 and A_2 *had the same demand*, they each get different rate allocations because the applications use different capacity allocation policies. Also note that all flows in the demand-limited application get the same rate; the sustainable network rate was below the 0.5 pkts/sec demanded by B_2. All flows experienced congestion from time 0 till about 600 seconds when all four sets of flows were active (a total of 74 flows). After set B_2 finished at 600 seconds, the other flows were allocated higher rates to take advantage of the increased available capacity.

Finally, Figure 16 shows the goodput achieved at the sink by each node. Two flows (for two different applications) from the same node are stacked together to show the total goodput achieved by each node. The average goodputs achieved by the two applications are 0.718 pkts/sec and 0.508 pkts/sec respectively, which totals 1.226 pkts/sec. This brings up an important point. The total achieved goodput is approximately 60% higher than the single-sink 40-node experiment (Section 4.2.1). This comes from using a tiered network. The two sinks are near the center of the network and roughly have comparably sized sub-trees. Moreover, the two sinks

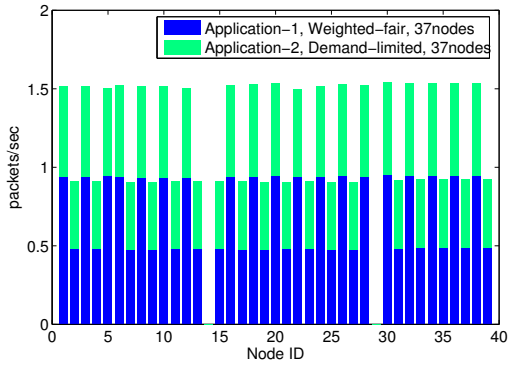

Fig. 16. Per-node goodput (two flows for same node stacked together) in 39-node experiment with two applications running on two sinks

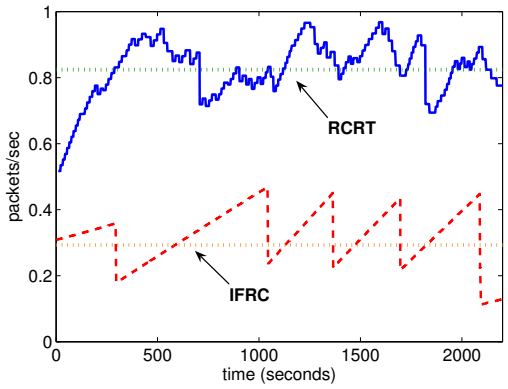

Fig. 17. Rate achieved by IFRC and RCRT in 30-node experiment

use 802.11b radios to communicate with each other, which has at least an order of magnitude higher bandwidth. This experiment not only shows that RCRT can support multiple concurrent applications on a tiered network with multiple sinks, but also quantifies the capacity increase achievable by using RCRT on a tiered network. Furthermore, although we have left inter-sink cooperative rate control as future work, this experiment shows that independent congestion control decisions made by different sinks resulted in reasonable (although not perfect) behavior.

4.2.5 *Comparison.* In this section, we compare the performance of RCRT with that of IFRC [Rangwala et al. 2006], a recently proposed interference-aware distributed rate-control protocol, and also with WRCP [Sridharan and Krishnamachari 2009], another recently proposed distributed rate-control protocol.

Figure 17 shows the rate achieved by IFRC together with RCRT's rate allocation in a 30-node experiment. The two protocols are qualitatively different, since IFRC does not provide end-to-end reliable transmissions. However, we are interested in

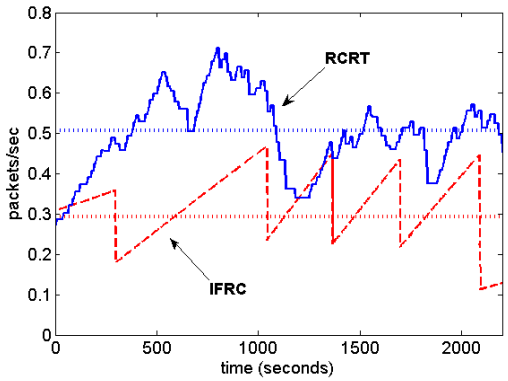

Fig. 18. Rate achieved by IFRC and RCRT in a 30-node experiment with modified MAC parameters for software acknowledgment

comparing their congestion control efficacy: IFRC does distributed congestion control, while RCRT performs congestion control at the sink, and we explore to what extent these approaches are quantitatively different. To compare these two protocols, we run them on the same set of nodes with the same radio power. RCRT was configured to use the fair allocation policy. However, IFRC has been evaluated only on static routing trees [Rangwala et al. 2006], so we ran IFRC on an empirically-determined "good" tree. [7] RCRT runs with dynamic routing enabled.

In Figure 17, the solid line represents the rate allocated to all nodes in RCRT, and the dashed line represents the rate achieved by one of the nodes in IFRC. Since all nodes were allocated the same rate in RCRT, a single line is sufficient to show rate allocation of *all* nodes. In IFRC, all nodes adapt their rates in near-synchrony, and rate plots for various nodes overlap with each other. We only plot the rate adaptation plot for one node for clarity. The dotted horizontal lines show the average rate achieved by each protocol during the experiment.

The results show that RCRT achieves an average rate of 0.824 pkts/sec in this experiment, which is more than twice the rate achieved by IFRC: 0.293 pkts/sec.

This surprisingly large difference in the result deserves further investigation. After extensive experiments, we have found that a hardware limitation of our current platform degrades the performance of IFRC. IFRC, by design, requires the radio and MAC to run in promiscuous mode and overhear the packets in the neighborhood. However, our experimental platform (CC2420) does not permit the use of hardware-level acknowledgments along with promiscuous mode. So the IFRC implementation uses software acknowledgments for link-level retransmission, which adds some software delays in the MAC layer and reduces IFRC throughput. When RCRT uses the same software acknowledgment implementation as IFRC, RCRT's average goodput is about 1.7 times that of IFRC (Figure 18).

[7]Our IFRC experiments were executed by the lead author of the IFRC paper. We ran dynamic routing in RCRT because the performance was similar to static and we wanted to emphasize the fact that RCRT works over dynamic routing where as IFRC does not; a significant deficiency of both IFRC and WRCP.

We believe two main reasons account for this difference. The first, of course, is that much of RCRT's performance advantage comes from implementing its congestion control functionality at the sink, which has a more global view of network state. This results in less pronounced rate deviations in RCRT's rate allocation profile. A second reason is that IFRC aggressively *avoids congestion* whereas RCRT *detects congestion* after it has happened. To avoid dropping packets, IFRC detects incipient congestion and aggressively cuts its rate when queues exceed a small threshold. On the other hand, RCRT fully utilizes the network queues until packets are lost, resulting in higher throughput. RCRT can afford packet loss, since it has a built-in loss recovery mechanism. That said, it should be possible to improve IFRC performance by varying its parameters and making it less aggressive in avoiding congestion, at the possible expense of lower end-to-end goodput.

Recently, we have also compared the performance of RCRT with that of WRCP [Sridharan and Krishnamachari 2009]. Since the code for WRCP was not publicly available at the time of this writing, we ran RCRT on the same 40-node topology as that in the WRCP paper and compared the RCRT results to the WRCP result in their paper [Sridharan and Krishnamachari 2009]. Specifically, on the same testbed (Tutornet [8]), we used exactly the same set of nodes, same radio power (5) and channel (26), and same static routing topology as in their topology. The result is that RCRT achieves an average rate of 0.84 pkts/sec, which is about 1.4 times the rate reported for WRCP: 0.6 pkts/sec. Moreover, if we can assume that RTT is roughly the double of end-to-end latency, then the latency achieved by RCRT (Table V) is no greater than that achieved by [Sridharan and Krishnamachari 2009]. The reasons for this difference are similar as for IFRC: in particular, WRCP aggressively avoids congestion by conservatively dividing the receiver capacity into the estimated active flow counts.

4.3 Network Topology

In Section 4.2.2, we have shown that the low per-node goodput (0.8 pkts/sec) in our baseline experiment (Section 4.2.1) is mainly a function of the topology, not our protocol. Also, we have already mentioned that RCRT can achieve up to 60 pkts/sec on average in a single-source single-sink network. In this section, we investigate how network size and topology affects the rate achieved by RCRT, which in turn shows how well RCRT adapts to different topologies. We used a 40-node network for our baseline experiment to enable comparison with IFRC and WRCP (Section 4.2.5). In this section, we report results from two other network sizes, 80 and 20 nodes. We also report result from a 40-node network with different node densities (by varying the RF transmit power) and network depth (by varying the number of master nodes). Note that RCRT achieved approximately fair goodput as well as 100% reliable packet delivery in all of these experiments.

4.3.1 *Network Size.* Table V summarizes the rate achieved by RCRT from various network sizes that we have experimented with. It also shows the estimated achievable rate for each topology. This is calculated using the methodology described in Section 4.2.2. Specifically, for the 80 node topology (Figure 19) we com-

[8]Tutornet: Tiered Wireless Sensor Network Testbed. (http://enl.usc.edu/projects/tutornet/)

network size	max hopcount	contention factor	estimated rate	achieved rate	% achieved	feedback overhead	RTT (msec)
80	7	171	0.35	0.34	97%	18.5%	390
40	6	72	0.83	0.80	96%	15.8%	325
20	3	27	2.22	2.11	95%	14.1%	229

Table V. Rate achieved by RCRT for various network sizes and topologies.

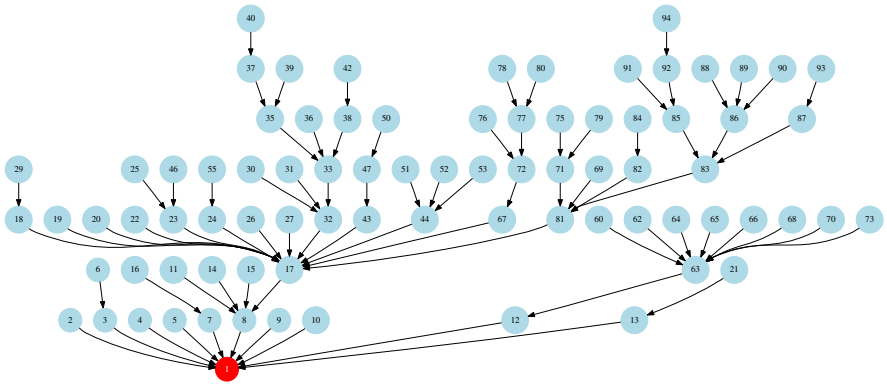

Fig. 19. A snapshot of routing topology constructed in a 80 node network

pute it as follows. In this topology, we estimate that node 17 is the most congested node: it has 54 children, 3 siblings and their children, and hence a parent (node 8) that has total of 58 children. Using the same calculation method as Section 4.2.2, the channel capacity around $17 \rightarrow 8$ must be shared *at least* by contention factor of 171 ($= 54 + 55 + 3 + 59$), even if we assume an ideal MAC. This means that the optimal sustainable rate in this network is $60/171 = 0.35$, which closely matches the achieved rate. The achievable rate estimate was calculated for the 20-node network in a similar manner. Thus, for three different network sizes, RCRT achieves between 95-97% of the achievable rate on the corresponding topology.

Table V also shows the average feedback overhead, measured as the total number of feedback packets divided by the total number of data packets, and the average RTT achieved by RCRT. Notice that even when the network size doubles from 40 to 80 nodes, the overhead and RTT increase is less than 20%. This is only a small increase. However, it is true that the overhead and latency will continue to increase if the network size continues to grow. However, with a larger network, the overall per-node throughputs will become vanishingly small, and it is not clear that such a configuration would be useful for the kinds of applications we have considered in this paper. Instead, in such cases, a tiered architecture with multiple sinks should be employed to reduce the feedback overhead and improve throughput as described below in Section 4.3.3.

4.3.2 *Network Density.* Next, we experiment with different network densities by varying the RF transmit power in a 40 node network. This effectively changes the node density and the routing depth of the network in a given physical area. Table VI summarizes the rate, as well as the feedback overhead and RTT, achieved by RCRT with varying RF transmit power. There are several things to note in

		RF Power configuration								
		3	**5**	**7**	**11**	**15**	**19**	**23**	**27**	**31**
dbm		-25	∼	-15	-10	-7	-5	-3	-1	0
hopcount	avg.	3.3	2.8	2.7	2.2	1.9	1.8	1.6	1.7	1.6
	max.	7	6	5	4	3	3	3	3	3
avg. goodput		0.81	0.95	0.97	0.98	0.98	1.07	0.96	1.10	1.06
avg. overhead (%)		15.1	15.0	15.5	15.9	15.2	15.2	16.5	15.5	16.0
avg. RTT (ms)		548	505	396	363	359	348	305	311	290

Table VI. RCRT results on 40-node network with varying RF power settings.

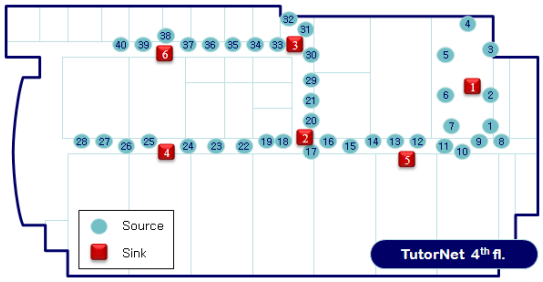

Fig. 20. Layout of source and sink nodes in Tutornet.

this result. First, as expected, the network depth decreases as the transmit power increases, and RTT is smaller when the network depth is smaller. Second, the feedback overhead stays at around 15% regardless of the network density. Finally, the rate achieved by RCRT is also fairly consistent even at different densities for comparable depths. The latter two results are because the congestion usually occurs near the sink and the total number of flows passing through that region governs the achievable rate of the network. On the contrary, as we will see below in Section 4.3.3, the rate and overhead achieved by RCRT will differ when the number of flows passing through the congestion region changes. Thus, increasing the transmit power within a given network is unlikely to improve the rate achieved by the network.

4.3.3 *Network Depth.* In this experiment, we vary the number of sinks in the network while keeping the number of nodes and RF power constant. Figure 20 depicts the layout of the source and sink nodes in our testbed where the darker squares represent the sink nodes with the numbering in their added order. Table VII summarizes the result from this experiment. The average and maximum hop count (routing depth) decreases as the number of sinks increase. Thus the average RTT also decreases. What is more important is that, unlike increasing the RF transmit power, the rate achieved by RCRT improves significantly and the overhead incurred also reduces as the number of sinks increase. This is because, as more sinks are added to the network, and the number of flows that a single sink handles reduces, and congestion is distributed among the several sinks. Thus, deploying additional sinks in a given network is an effective way of improve the throughput, as well as overhead and latency, achieved by RCRT.

		Number of Master(Sink) nodes					
		1	**2**	**3**	**4**	**5**	**6**
hopcount	avg.	2.8	1.6	1.5	1.4	1.3	1.0
	max.	6	3	3	3	3	2
avg. goodput		0.87	1.33	1.62	2.32	2.56	2.78
avg. overhead (%)		15.8	14.6	15.1	13.9	13.3	13.5
avg. RTT (ms)		492	256	208	194	189	162

Table VII. RCRT results on 40-node network with varying number of sink nodes.

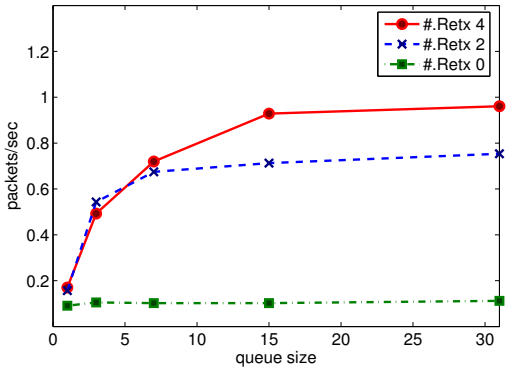

Fig. 21. Goodput achieved by RCRT in the 40-node experiment with varying queue sizes and maximum number of hop-by-hop retransmissions.

4.4 Sensitivity: Hop-by-hop Retransmissions and Queue Sizes

As we have mentioned earlier (Section 3.2), the design of RCRT does not depend on any features specific to a particular MAC or routing layer. We only assume that there exists at least one or more end-to-end path, with non-zero delivery probability, from each sensor node to the sink and also a reverse path from the sink to each node. However, as we have mentioned in Section 3.5 also, the performance of RCRT's rate adaptation may depend on the quality of these end-to-end paths. If the end-to-end packet delivery ratio is extremely poor, RCRT may over-estimate congestion and assign lower rates than optimal.

In this section, we investigate the effects of lower-layer parameters to the performance of RCRT. Specifically, we look at the effects of the limit on the number of hop-by-hop retransmissions (the *retransmission limit*) at the MAC layer and the size of the forwarding queue at the routing layer, since these two factors may impact path packet loss characteristics.

Figure 21 shows the goodput achieved by RCRT in the 40-node experiment, with varying queue sizes and varying retransmission limits. The bottom dash-dot line (with squares), the dotted line (with crosses), and the solid line (with circles), each show the goodput achieved by RCRT when the retransmission limits are 0, 2 and 4, respectively. There are several observations that we can make from this figure. First, RCRT achieves very low goodput (∼0.1 pkts/sec) regardless of the queue size when there are no hop-by-hop retransmissions. On the other hand when the retransmission limit is either 2 or 4, the queue size impacts the rate achieved by

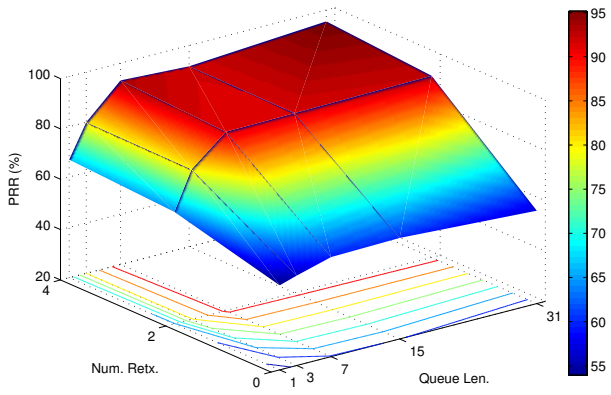

Fig. 22. PRR achieved by best-effort transport (z-axis) with varying forwarding queue sizes (x-axis) and varying maximum number of hop-by-hop retransmissions (y-axis).

RCRT. This is because, when there are no hop-by-hop retransmissions, successful packet delivery probability is so poor that RCRT assigns low rates and the number of packets in the network is very small, so the forwarding queue never fills up.

A second observation requires a more careful understanding. When the forwarding queue size is above a certain point (15 in this figure), a retransmission limit of 4 achieves better goodput compared to that of 2. However, when the queue size is below a certain point (7 in this figure), retransmission limits of 2 and 4 have similar goodput. This is because more hop-by-hop retransmissions result in higher utilization of the forwarding queue and possible queue drops. In general, more hop-by-hop retransmissions should result in better end-to-end packet delivery ratio, and hence better goodput achieved by RCRT. However, when the queue sizes are small, a higher rate results in more queue drops, which in turn takes away the extra packet delivery achieved by more hop-by-hop retransmissions. Thus more hop-by-hop retransmissions do not improve the goodput achieved by RCRT when forwarding queue sizes are too small.

To verify the above arguments, we ran a series of experiments to measure the end-to-end packet reception ratio (PRR) achieved by the best-effort transport in the same 40 node network with varying forwarding queue sizes and varying retransmission limits. Figure 22 is a 3-D plot of all the results: the z-axis represents the PRR achieved by the best-effort transport, and the x and y axes plot forwarding queue sizes and retransmission limits, respectively. Observe that when there are no hop-by-hop retransmissions, the PRR is very poor regardless of what queue size we use. However, when there are hop-by-hop retransmissions (limit of 2 or 4), queue size does affect PRR. Another observation is that, when the queue size is very small (1, the minimum possible), the PRR is very poor even with hop-by-hop retransmissions. However, when the queue size is sufficiently large relative to the network size, hop-by-hop retransmissions can significantly improve end-to-end PRR. These results confirm that our reasoning about Figure 21 is valid.

In summary, the performance of RCRT may depend on the end-to-end packet

	Additive Increase Parameter 'A'							
	0.25	**0.5**	**1**	**2**	**4**	**5**	**10**	**20**
avg. goodput	0.72	0.91	0.93	0.99	0.95	0.62	0.66	0.56
avg. overhead (%)	18.5	16.5	15.6	14.5	16.8	31.2	28.4	34.0
avg. RTT (ms)	331	390	514	495	662	551	485	428

Table VIII. RCRT results on 40-node network with varying additive increase parameter 'A'.

delivery ratio of the source nodes in the network. Hop-by-hop retransmissions at the MAC layer and the queue size at the routing layer are two important factors that may affect the end-to-end packet delivery. However, having a moderate queue size (15 packets) and reasonable retransmission limits (4) result in good RCRT performance, across all our experiments.

4.5 Additive Increase Parameter 'A'

In this section, we investigate the effect of the value of the additive increase parameter A on the performance of RCRT. As we have discussed in Section 3.5, RCRT additively increments the total aggregate rate $R(t)$ by A when it observes the network as under-utilized. If this A is too small, it will take long time for RCRT to throttle its rate up to the steady-state level whenever the rate is below it. If this A is too large, RCRT's rate adaptation will oscillate significantly around the steady-state level. Thus, setting the value of A incorrectly can result in significant performance loss. For this reason, we explore the sensitivity of RCRT's performance to the choice of A.

Table VIII summarizes the average rate, overhead, and RTT achieved by RCRT in a 40-node experiment with varying A. There are several observations that can be made from this result. First, when A is too large (5 or higher), RCRT incurs high overhead and achieves lower rate due to the oscillatory behavior of its rate adaptation. This is because it increases the rate quickly and decreases frequently, instead of spending more time near the steady-state rate (between the upper and the lower threshold). Second, when A is too small (0.25 or lower), RCRT achieves lower overall goodput because it takes long time for it to recover its steady-state rate after a congested period. Note that this does not necessarily worsen the overhead and RTT. Finally, RCRT achieves more or less comparable rates for a wide range of A values, between $0.5 \sim 4$. In this range, the average overhead and average RTT are also comparable. Thus, the result altogether shows that it is possible for RCRT to select any value of A within this range; in our other experiments, we have used a conservative value of $A = 0.5$.

5. REAL-WORLD DEPLOYMENT

In this section, we discuss a real-world deployment of RCRT, used in an imaging application for bird nestbox monitoring at the *James San Jacinto Mountain Reserve*[9] [Hicks et al. 2008]. This application used RCRT to reliably transfer images of bird nestboxes in a large mountain area for three months.

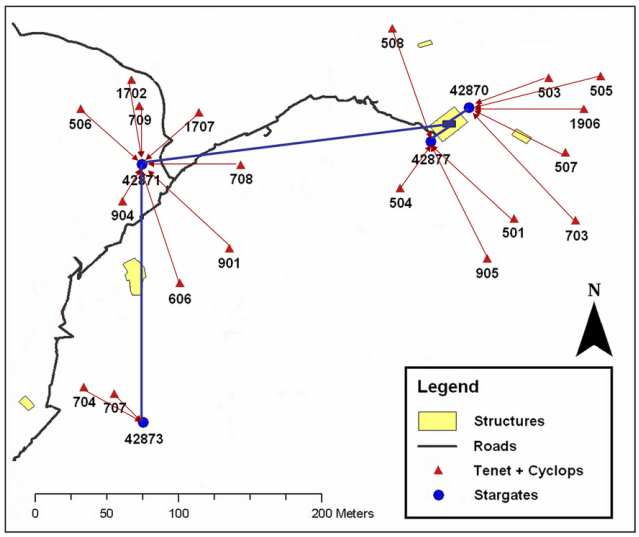

Fig. 23. A map detailing the location and topology of our deployment at *James Reserve*.

5.1 Motivation and Deployment

Studies have been taking place for several years at the *James San Jacinto Mountains Reserve* on the breeding biology of some species of birds. These studies involve observing the behavior of birds in *nest boxes*, specially placed to attract birds. However, observing day-to-day changes in breeding behavior in these nest boxes is extremely labor intensive. Each box spread over a large mountain area must be checked daily by a biologist in the field. A wireless sensor network with image sensors can help: the biologists can inspect the wirelessly delivered images remotely, saving time and effort. For this purpose, we developed a wireless imaging application on TENET [Paek et al. 2010] using the RCRT protocol to transfer images periodically taken by Cyclops [Rahimi et al. 2005] cameras. We deployed this system, consisting of 5 master nodes and 19 motes, over 120,000 square meters of the James Reserve, for 3 months between May 9th $\sim$ Aug 9th, 2008.

The goal of our application was to repeatedly collect, from every node, an image along with environmental sensor readings as frequently as possible. Reliable delivery is a requirement for our system, otherwise image quality can be severely compromised. Each image is large (40 KB uncompressed), and the total network data rate required to deliver images from all our nodes can easily overwhelm the radio bandwidth. Our system uses lossless image compression to alleviate this somewhat, but congestion control, which allows our application to adapt its image transfer rate dynamically, is still necessary to avoid congestion collapse. It is also necessary to achieve efficiency in the presence of network dynamics: in our deployment, some nodes ran out of battery, and were unattended for a while, after which the batteries were replaced and they rejoined the network; also, the wireless environment changed as foliage grew during the 3-month period. For these reasons,

[9] `http://www.jamesreserve.edu/`

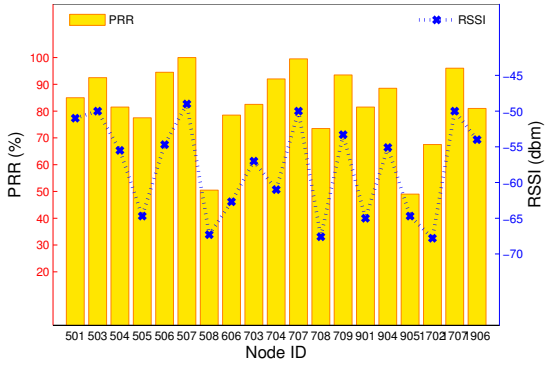

Fig. 24. Link connectivity of each node: Packet reception rate and RSSI to next hop node.

we used RCRT as our transport protocol in this application.

In our deployment, we had one Linux server machine running as the RCRT sink, and this server communicated with four Stargates which were placed around the James Reserve to constitute the master tier of the network. The server and the Stargates were connected via Ethernet and 802.11b wireless. Each Stargate was connected to a Mica2 mote that acted as gateway to nearby sensor nodes. Each sensor node was a Mica2 mote equipped with a Cyclops camera. The resulting network topology is shown in Figure 23. While our system does support multihop routing (and we did log some temporary multihop paths), nest box placement was determined from a previous, single-hop deployment [Ahmadian et al. 2008].

5.2 Results

During the 3 month deployment period, RCRT delivered over *83 million* packets to collect 102,173 images from 19 sensor nodes. Since there were times when our application was stopped for maintenance or debugging purposes, our networking log files are discontinuous. Here we present the observations we made for one week from July 21 to 28, 2008. During this period, there were total of 8453 attempts to transfer an image from 19 nodes. Among these attempts, 8372 image transfers, which corresponds to 99% of initiated RCRT transfers, were complete with 100% packet delivery. For these complete images, the average data rate achieved by the network was 1.1 packets/sec per node, and the average number of packets required to deliver one image was 833.2. As a result, one image transfer took 12.6 minutes on average (each image is 40KB, but had a variable size after compression, so there was some variability in the number of packets required to deliver an image).

This data rate of 1.1 pkts/sec was achieved despite extremely poor link connectivity. Figure 24 shows the end-to-end packet delivery ratio for each node when we tested the network just before the deployment using best-effort delivery without loss recovery. (e.g. node 905 had PRR of less than 50%). The figure also plots the RSSI readings to nearest routing parent for each node. When we conducted a single-source single-sink experiment with 433MHz Mica2 motes, we found that

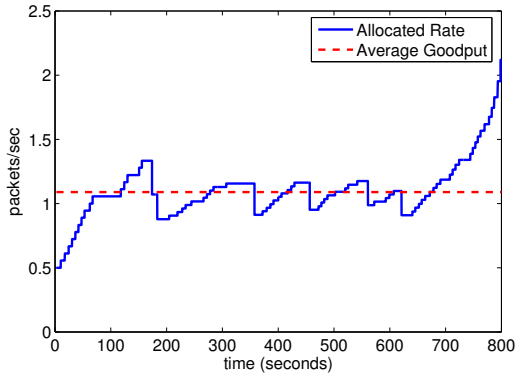

Fig. 25. Rate-allocation profile at James Reserve for single image transfer

RCRT can achieve up to 11 pkts/sec on average. [10] From Figure 23, notice that Stargate 42871 at the top-left area of the map is the bottleneck where 8 sensor nodes are trying to route through a single stargate. This means that the maximum achievable rate in this topology cannot exceed $11/8 = 1.375$ pkts/sec even with an ideal MAC when fair rate allocation policy is used. Considering the extremely poor link connectivities (Figure 24) and the fact that RCRT may over-estimate congestion when PRR is very low (Section 3.5), the deployment result of 1.1 pkts/sec (which is 80% of 1.375) is very good.

The reason that only 99% of the initiated RCRT transfers were complete is as follows. Poor link connectivity for some nodes caused frequent packet losses and delayed loss recovery severely. But our *application* terminated an image transfer if no new image segment came in within 30 seconds. So whenever there were lost packets near the end of image transfers that were not recovered within 30 seconds, our application stopped without waiting for the RCRT loss recovery to complete. Given more time, RCRT would have been able to correctly transfer those images.

Figure 26 is a snapshot of the rate allocation profile for 10 consecutive image transfers during a 2 hour 45 minute period. Also, Figure 25 is a zoomed-in version of Figure 26 which plots the rate allocated to each node for a single image transfer during 13 minute period. There are several things to notice here. First, the average data rate was consistently around 1.1 pkts/sec most of the time for most of the image transfers. It is visually evident that RCRT's rate adaptation stayed near the steady-state average by making small rate reductions. Second, the reason that the rate starts at 0.5 pkts/sec at the beginning of each image transfer is because this was set as the initial value for our AIMD rate adaptation. (We start a new RCRT connection for every image transfer). Finally, the rate shoots up to around 2.5 pkts/sec near the end of each image transfer. This is because there are multiple nodes in the network and some nodes finish data transfer earlier than others due to the variability in loss rates (Figure 24). When subset of the nodes finish transfer earlier than others, there is left-over network capacity, which RCRT detects and

[10] We use an 80 byte TinyOS packet payload.

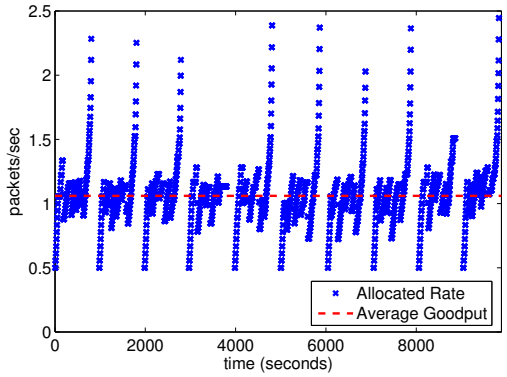

Fig. 26. Rate-allocation profile at James Reserve for 10 image transfers (2 hour 45 minute period)

adapts to, so the incomplete transfers are allocated higher rates as more nodes complete.

In summary, we have used RCRT for three months in a real-world deployment of 19-node network deployed over an area of 120000 square meters (Figure 23). During this deployment, RCRT collected over 83 million packets reliably at a reasonable rate despite highly variable individual node availability and wireless link quality. Our deployment have shown that RCRT works well in real deployments.

6. CONCLUSIONS AND FUTURE WORK

RCRT is a reliable transport protocol for wireless sensor networks. It places its congestion control functionality at the sink, whose perspective into the network enables better aggregate control of traffic, and affords flexibility in rate allocation. It supports multiple concurrent applications, and is robust to network dynamics. Finally, RCRT's rates are significantly higher than that of the state-of-the-art in sensor network congestion control. We envision several interesting directions for future work including coordinated rate allocation across sinks, differential treatment for flows unconstrained by the bottleneck region, and improved convergence time in networks with highly varing RTTs.

REFERENCES

Ahmadian, S., Ko, T., Hicks, J., Rahimi, M., Estrin, D., Soatto, S., and Coe, S. 2008. Heartbeat of a nest: using imagers as biological sensors. *In submission to ACM Transactions on Computational Logic*.

Ali, A. M., Yao, K., Collier, T., Taylor, C., Blumstein, D., and Girod, L. 2007. An empirical study of collaborative acoustic source localization. In *Proc. of the IPSN/SPOTS*.

Balakrishnan, H., Rahul, H. S., and Seshan, S. 1999. An integrated congestion management architecture for internet hosts. *SIGCOMM Comput. Commun. Rev. 29,* 4, 175–187.

Bian, F., Rangwala, S., and Govindan, R. 2007. Quasi-static centralized rate allocation for sensor networks. In *Proceedings of IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*.

Caffrey, J., Govindan, R., Johnson, E., Krishnamachari, B., Masri, S., Sukhatme, G., Chintalapudi, K., Dantu, K., Rangwala, S., Sridharan, A., Xu, N., and Zuniga, M. 2004.

Networked sensing for structural health monitoring. In *Proceedings of the 4th International Workshop on Structural Control*.

Chintalapudi, K., Paek, J., Gnawali, O., Fu, T., Dantu, K., Caffrey, J., Govindan, R., and Johnson, E. 2006. Structural damage detection and localization using netshm. In *Proceedings of IPSN/SPOTS'06*.

Chiu, D.-M. and Jain, R. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst. 17,* 1, 1–14.

Eggert, L., Heidemann, J., and Touch, J. 2000. Effects of ensemble-tcp. *ACM Computer Communication Review 30,* 1 (Jan.), 15–29.

Floyd, S. 2000. Congestion control principles. RFC2914.

Floyd, S., Handley, M., Padhye, J., and Widmer, J. 2000. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM Conference*.

Floyd, S. and Jacobson, V. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1,* 4, 397–413.

Floyd, S., Jacobson, V., Liu, C.-G., McCanne, S., and Zhang, L. 1997. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking 5,* 6, 784–803.

Gnawali, O., Fonseca, R., Jamieson, K., Moss, D., and Levis, P. 2009. Collection tree protocol. In *Proceedings of 7th ACM International Conference on Embedded Networked Sensor Systems (SenSys'09)*.

Hicks, J., Paek, J., Coe, S., Govindan, R., and Estrin, D. 2008. An easily deployable wireless imaging system. In *Proceedings of ImageSense08: Workshop on Applications, Systems, and Algorithms for Image Sensing*.

Hui, J. and Culler, D. 2004. The dynamic behavior of a data dissemination algorithm at scale. In *Proceedings of 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys'04)*.

Hull, B., Jamieson, K., and Balakrishnan, H. 2004. Mitigating congestion in wireless sensor networks. In *Proceedings of 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys'04)*.

Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. 2002. Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking 11,* 1 (February), 2–16.

Iyer, Y., Gandham, S., and Venkatesan, S. 2005. Stcp: a generic transport layer protocol for wireless sensor networks. In *Proceedings of International Conference on Computer Communications and Networks*.

Jacobson, V. 1988. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM Conference*.

Katabi, D., Handley, M., and Rohrs, C. 2002. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*.

Kim, S., Fonseca, R., Dutta, P., Tavakoli, A., Culler, D., Levis, P., Shenker, S., and Stoica, I. 2007. Flush: A reliable bulk transport protocol for multihop wireless networks. In *Proceedings of 5th ACM International Conference on Embedded Networked Sensor Systems (SenSys'07)*. 351–365.

Li, M., Agrawal, D., Ganesan, D., and Venkataramani, A. 2009. Block-switched Networks: A New Paradigm for Wireless Transport. In *Proceedings of 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. USENIX.

Musaloiu-E., R., Liang, C.-J. M., and Terzis, A. 2008. Koala: Ultra-low power data retrieval in wireless sensor networks. In *Proceedings of 7th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'08)*.

Paek, J., Chintalapudi, K., Cafferey, J., Govindan, R., and Masri, S. 2005. A wireless sensor network for structural health monitoring: Performance and experience. In *Proceedings of The Second IEEE Workshop on Embedded Networked Sensors, (EmNetS-II)*. 1–10.

Paek, J., Greenstein, B., Gnawali, O., Jang, K.-Y., Joki, A., Vieira, M., Hicks, J., Estrin, D., Govindan, R., and Kohler, E. 2010. The tenet architecture for tiered sensor networks. *ACM Transactions on Sensor Networks 6,* 4.

Rahimi, M., Baer, R., Iroezi, O. I., Garcia, J. C., Warrior, J., Estrin, D., and Srivastava, M. 2005. Cyclops: In situ image sensing and interpretation in wireless sensor networks. In *Proceedings of 3th ACM International Conference on Embedded Networked Sensor Systems (SenSys'05)*. 192–204.

Ramakrishnan, K. K. and Jain, R. 1990. A binary feedback scheme for congestion avoidance in computer networks with connectionless network layer. *ACM/IEEE Transactions on Networking 8,* 2, 158–181.

Rangwala, S., Gummadi, R., Govindan, R., and Psounis, K. 2006. Interference-aware fair rate control in wireless sensor networks. In *Proceedings of ACM SIGCOMM Symposium on Network Architectures and Protocols*.

Sankarasubramaniam, Y., Akan, O. B., and Akyildiz, I. F. 2003. Esrt: Event-to-sink reliable transport in wireless sensor networks. In *Proceedings of 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '03)*. ACM, 177–188.

Sridharan, A. and Krishnamachari, B. 2009. Explicit and precise rate control for wireless sensor networks. In *Proceedings of 7th ACM International Conference on Embedded Networked Sensor Systems (SenSys'09)*.

Stann, F. and Heidemann, J. 2003. Rmst: Reliable data transport in sensor networks. In *Proceedings of 1st IEEE Workshop on Sensor Network Protocols and Applications (SNPA)*.

Stathopoulos, T., Girod, L., Heidemann, J., and Estrin, D. 2005. Mote herding for tiered wireless sensor networks. Tech. Rep. 58, CENS. Dec. 7.

USC/ISI. 1981. Transmission control protocol. RFC793.

Wan, C. Y., Campbell, A. T., and Krishnamurthy, L. 2002. Psfq: A reliable transport protocol for wireless sensor networks. In *Proceedings of 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*. ACM.

Wan, C.-Y., Eisenman, S. B., and Campbell, A. T. 2003. Coda: Congestion detection and avoidance in sensor networks. In *Proceedings of 1st ACM International Conference on Embedded Networked Sensor Systems (SenSys'03)*.

Xu, N., Rangwala, S., Chintalapudi, K., Ganesan, D., Broad, A., Govindan, R., and Estrin, D. 2004. A wireless sensor network for structural monitoring. In *Proceedings of 2nd ACM International Conference on Embedded Networked Sensor Systems (SenSys'04)*.

Zhang, H., Arora, A., Choi, Y., and Gouda, M. 2003. Reliable bursty convergecast in wireless sensor networks. In *Proceedings of 4th ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '03)*. ACM.