ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks»

ΗΛΙΑΣ ΠΑΙΔΑΚΑΚΟΣ

MM4120011

ΑΘΗΝΑ, ΟΚΤΩΒΡΙΟΣ 2014

Abstract

This thesis [1] focuses on examining a rate-controlled reliable transport protocol (RCRT) suitable for constrained sensor nodes that concurrently transmit large volume of data used by high-bitrate applications (imagers, accelerometers, microphones). These applications are also loss-intolerant. RCRT uses end-to-end explicit loss recovery, but places all the congestion detection and rate adaptation functionality in the sinks. Sink-driven congestion control and rate adaptation has great efficiency on supporting traffic flows in highly unpredictable sensor network environments. It can avoid sink implosion, since it has a comprehensive view of network behavior (on the sink).

Our environment consists of multiple sensors interconnected by a multi-hop ad-hoc network and at least one sink (we can also support multiple sinks). We examine scenarios where sensor and sink are fixed, taking also into consideration the multiple classes of flows occurring inside the network. Our approach of controlling congestion is to perform both receiver-based and in-network rate control, based on the current network condition as a whole, the delay requirements of incoming packets, rejecting those packets with higher probability of missing their deadlines. Finally, we assume that an underlying routing protocol is available to forward packets from the sources to the sink and ensure node connectivity, although we make no assumptions about the type of the routing protocol.

Thus, the present thesis focuses on examining the RCRT protocol, its goals and the components that describe it in the most appropriate way.

1. Introduction

A sensor network consists of multiple detection stations, called sensor nodes, which are extremely small, have low power, low cost and can communicate over a wireless topology using radio frequencies or optical wireless. Nodes collect environmental data which they then forward to an infrastructure processing node. The kinds of phenomena that can be sensed is acoustics, light, humidity, temperature, imaging, seismic activity, and in general any physical phenomena that will cause a transducer to respond. As Intel announced, "Someday, billions of embedded chips and sensing devices will be integrated into objects and locations that are part of our daily lives including clothes, cars, etc."

Deployment and applications of wireless sensor nodes is crucial as a way to monitor and control systems consisting of thousands of nodes in high densities, which is extremely demanding. This can be noted by the purpose of these applications such as monitoring and warning natural of disasters and their effects (floods, winds and hurricanes), protecting homeland in terms of security considerations, conducting military surveillance, environment changing and health monitoring.

Wireless sensor networks are characterized by constraints on available power resources, processing speed, storage capacity, communication bandwidth and the harsh environment on which they function. In order to effectively use sensor networks, we must minimize both computation costs and communication costs at the sensors. Computation costs involve rate adaptation, rate allocation and congestion control, while communication costs involve the transmission and retransmission of data and control packets.

To support high-rate applications transferring large amount of data, we need to address two major problems. On the one hand, we have to take into account that sensors which have limited radio bandwidth are likely to easily overwhelm the network until congestion collapse occurs. On the other hand, applications that use high-rate sensors are often loss-intolerant. While other sensor network transport protocols solve one of the two problems, RCRT solves both these two requirements: Firstly, the network must be able to support multiple concurrent streams from each sensor and secondly it separates the traffic allocation policies from the transport mechanisms.

Additionally, in-network processing is used to aggregate information from various sensors and to summarize this particular information before communication establishment and passing information to other nodes. It is of high importance for wireless networks to be robust because of the dynamic conditions under which they operate and maximize the system lifetime as whole, rather than for the individual nodes themselves. Due to the limited network capacity, the small queue sizes and the restricted processing capabilities of the nodes, we use localized algorithms to prevent single points of failure.

2. RCRT Overview

RCRT [2] is a transport layer protocol composed of a series of mechanisms/components, summarized in Table 1. First of all, we define a sink as an entity which runs on a base station and which collects data from one or more sensors. Data sourced by the sensors can be raw samples, temperature indicators, images, videos and so on. With RCRT we have the capability to have more than one sink running concurrently in the sensor network, but due to limitations we examine the behavior of one sink. We use the notation ***fij*** to denote the flow of data from source *i* for sink *j*. This flow can be delivered from the source over multiple hops to the base station. Finally, each sink *j* is associated with a capacity allocation policy ***Pj*** which determines how network capacity is divided up across different flows ***fij*** for every sensor ***i***. The simplest ***Pj*** is one in which each flow ***fij*** gets an equal/fair share of the network capacity. We examine this later at Rate Allocation Chapter where the different allocations policies are described at a more detailed way.

RCRT provides reliable, sequenced delivery of flows ***fij*** from source *i* to sink *j*. Furthermore, RCRT ensures that, for a given application *j*, the available network capacity is allocated to each flow according to policy ***fij***. Specifically, each flow *fi j* is allocated a rate ***rij(t)*** at each instant *t* that is in accordance with policy ***Pj***. Thus, for a fair allocation policy, all sensors would receive equal ***rij(t)***.

Function	Where	How
End-to-end retransmissions	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 1: The various components of RCRT.

Traffic management functionality resides at the sink. 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.

End-to-end reliability is achieved using end-to-end negative acknowledgments (NACKs). Specifically, each sink determines congestion levels and makes rate allocation decisions. Once the sink *j* decides the rate ***rij***, it piggybacks this rate in two ways; either with a negative acknowledgment packet, or via a separate feedback packet, to source *i*.

RCRT has three distinct logical components at the sink. The congestion detection component observes the packet loss and recovery (which packets have been lost, how long it takes to recover a loss) across every flow ***fij***, and decides if the network is congested. Once the network is congested, the rate adaptation component estimates the total sustainable traffic ***R(t)*** in the network since it has the whole network overview. Then, the rate allocation component decreases the existing flow rates ***rij(t)*** to achieve ***R(t)***, while conforming to policy of each flow ***Pj***. Conversely, when the network is not congested, the rate adaptation component additively increases the overall rate ***R(t)***, and the rate allocation component determines ***rij(t)*** to exploit maximum network performance.

Finally, RCRT focuses on the transport protocol itself. Although the link layer and the routing layer are largely consistent with RCRT, its design does not depend on any features specific to a particular MAC layer. 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.

3. Design Goals

RCRT is designed for sensor networks in which sensor reading is transmitted from one or more sources to a base station. This design manages to achieve the following goals.

End-to-end-reliability

The first goal is to achieve complete end-to-end reliability of all data transmissions by each sensor to the sink. Examples of applications such as imaging, acoustic source localization and structural health monitoring emerging high data rate exchange and are also loss in-tolerant. In these cases packet loss may affect the accuracy of processing algorithms.

Network Efficiency

The second goal is to achieve network efficiency since on these high rate applications packets are repeatedly lost and continuously retransmitted. Specifically, we need to avoid congestion collapse while maintain the highest possible data transmission from source (sensors) to base (sink).

Support for concurrent applications

Sensor network deployments are already evolved on multi-user or multi-application systems; this is the purpose of supporting multiple concurrent (temperature, fire, photo, video, etc).

Flexibility

Different applications are more demanding in terms of data aggregation and correlation on the sink as well as higher rates of data exchange. RCRT has the ability to spread different allocation capacity policies to different applications due to criticality mission and also the specifications and restrictions of each.

Minimal Sensor Functionality

Due to the constraints of the current generation of sensors, it is mandatory to move as much of the intelligence as possible out of the sensor network to the sink. Limited energy capacity of sensors must be taken into consideration in order to keep the protocol functionality down to 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 location of congested nodes. The system must be able to dynamically adapt to these changes.

4. Protocol Description

Communication between the sensors and the sink follows the protocol developed for the Distributed Sensor Systems for Emergency Response (DISFER) [3] project, and proceeds in five stages: connection establishment, sensor information exchange, data exchange, idle and connection release. This allows us to reuse the existing codebase for the DISFER protocol. On the other hand, as mentioned in section 6, there are significant differences on the congestion control, rate allocation and rate adaptation mechanisms between RCPT and DISFER, therefore we have extended the codebase with the corresponding parts for RCPT.

4.1. Connection Establishment

When the sink begins operation, it listens to a well-known UDP port for connection requests from sensors. The sensors wait until the sink becomes reachable before connecting to it. In DISFER prototype, the sink periodically sends a probe message, MSG_HELLO, to the well-known IP address and UDP port of the sink, until it receives a response; the probe interval is configurable. If the routing protocol supports it, the sink can alert the routing engine to be notified when the sink becomes reachable. The MSG_HELLO message includes the sensor identifier, unique in the sensor network, and its type; in our prototype, only event and continuous sensor types exist. When the sink receives such a message, it responds with an MSG_HELLO_ACK message which indicates a separate UDP port where the sensor should send the following control messages. The sink notes the time when it sent the message, with the purpose of measuring the round-trip time (RTT) to that sensor. The use of a separate control port per sensor allows the sink to dedicate one thread for receiving new connection requests, and a separate thread for each connection to a specific sensor, thus avoiding the need to multiplex messages from multiple sensors over a single control connection. The connection establishment messages are depicted in Figure 1.

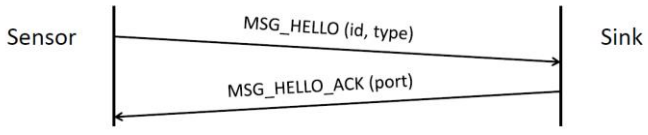

Figure 1: Connection Establishment stage.

4.2. Sensor Information and Idle

After receiving a response from the sink, the sensor prepares an MSG_INFO message which includes the delay between receiving the message from the sink and sending the response, the data rate requested by the sensor, whether the sensor is ready to send data at this point in time or not, the data packet size and the total size of the data to send. When the sink receives the MSG_INFO message it calculates the time elapsed since sending the SG_HELLO_ACK, subtracting the delay in the message to get the RTT to the sensor. At this point, the connection has been established and the sink is aware of the sensor's bandwidth requirements. If the sensor indicates that it is not ready to send data at this point, the sink will moves to an idle state, waiting until the sensor sends a MSG_CTRL_DATA with no parameters. Then, the sink starts the data exchange by sending an MSG_CTRL_START message to the sensor, indicating the data rate to use and a UDP data port to use for the transmission. If the sensor indicated in the MSG_INFO message that it was ready to send data, the MSG_CTRL_START message is sent immediately as a response. The sensor information messages are shown Figure 2.

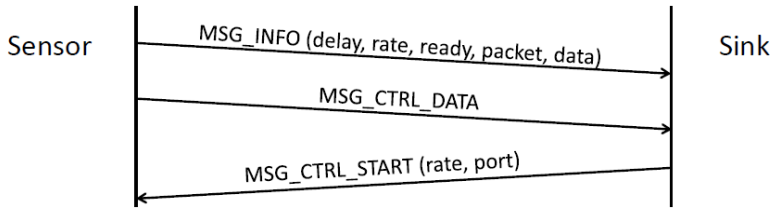

Figure 2: Sensor information exchange stage.

4.3. Data Exchange

After the sink sends the MSG_CTRL_START, the actual data transfer begins, using the UDP data port assigned for the transfer; control messages, such as NACKs and rate updates, are exchanged out of band over the control channel, without being rate controlled. This allows control messages to be sent without waiting behind a quite long, queue of data messages. The sensor breaks down its transmission into packets with the size indicated in the MSG_INFO message, until all the data indicated in the MSG_INFO message is exhausted. Data packets only have a single header field, a segment number used to sequentially number all data packets.

When a missing packet is detected, the sink sends a MSG_NACK to the sensor over the control channel. The sensor then, immediately retransmit lost messages. After the transmission is completed, the remaining packets from initial flow start to be transmitted again, generating further NACKs from the sink, if needed. This procedure is repeated in rounds, until all messages are received (Chapter 9 – Differences between DISFER and RCRT). Once recovery is complete, the sink sends a MSG_CTRL_DONE message to indicate a successfully completed data transfer. Both endpoints then move to an idle state, until the sensor generates a new MSG_CTRL_DATA message. If the need arises, the sink will send a MSG_CTRL_RATE message to the sensor indicating its new rate allocation. The data messages are shown in Figure 3.

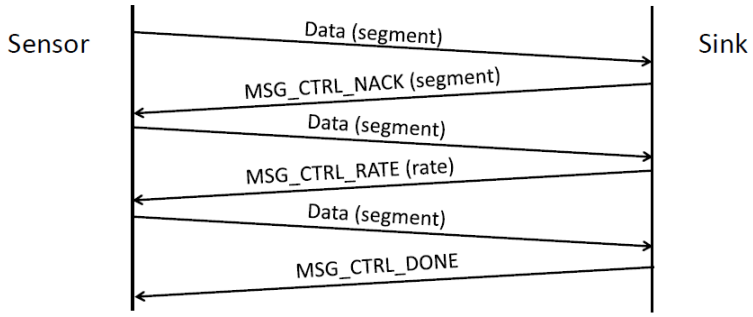

Figure 3: Data exchange stage.

4.4. Connection Control and Release

Since the path between the sink and the sensor may become disconnected due to the sink's mobility, the connection may fail between data transfers, without either side noticing. For this reason, the sink periodically sends an MSG_CTRL_ALIVE message to the sensor, which is acknowledged by an MSG_CTRL_ACK message from the sensor that includes the delay incurred between receiving the MSG_CTRL_ALIVE and responding with the MSG_CTRL_ACK. In addition to confirming that the connection is still alive, this procedure allows the sink to periodically measure the RTT of the connection. If the sensor or the sink requires the completion of the connection, they can send a MSG_CTRL_BYE message, which does not need to be acknowledged, as after either side drops the connection, the other one will eventually timeout: the sink times out if no responses are received to its MSG_CTRL_ALIVE messages, while the sensor times out if no MSG_CTRL_ALIVE messages arrive. The MSG_CTRL_BYE message simply

allows the other end to release the resources dedicated to the connection without waiting for a timeout. The connection control messages are shown in Figure 4.

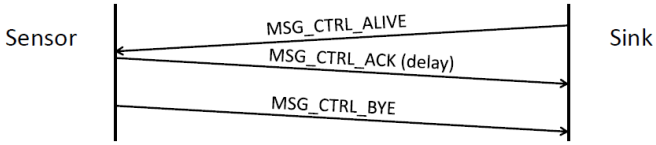

Figure 4: Connection control and release stage.

5. RCRT Components

5.1. End-to-End Reliability

RCRT uses NACK packets to ensure the reliability of data delivery. When the sink detects that a single packet or a number of packets have been lost, it immediately requests the retransmission for the specific packet/s from the sensor. Each sensor has a retransmit buffer that stores a local copy of each packet being sent. In order to avoid filling up the retransmission buffers, the sink can acknowledge the recently arrived packets, sending a cumulative-ACK (C-ACK). The sensor can then discard all copies of C-ACK and backwards packets from its buffer.

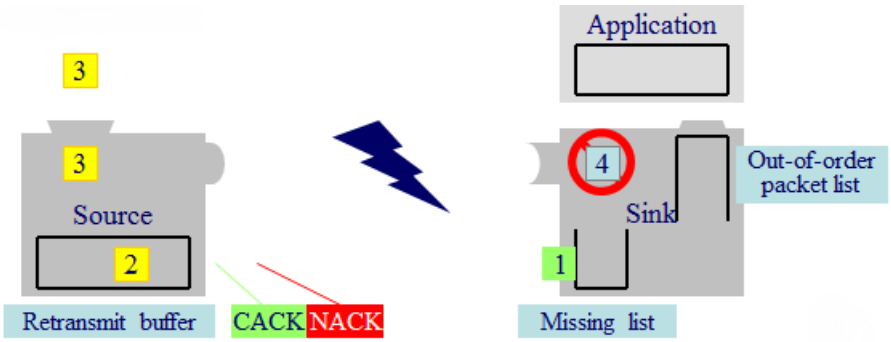

Figure 5: Overview of Lists for end-to-end reliability mechanism

The sink has an overview of incoming packets since it holds the sequence number of all packets of each flow in order to find a gap; this indicates a packet loss. The sink holds two lists to confront packet loss; one missing-Packet-List which stores the sequence number of missing packets need to be retransmitted from sensor and one out-Of-Order-List which holds the out-of-order packets before they depart to application layer. When the missing packet/s are retransmitted, then the out-of-order packets are delivered directly to the application layer. We have to make a notice that out-Of-Order-List is being used also for congestion detection. We will explain this in a more detailed way on the next chapter.

5.2. Congestion Detection

RCRT attempts to measure the network congestion at the sink. This approach is closer to what wired networks do; that have attempted to detect congestion at the ends [4, 5]. There is one

important difference however: 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.

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 approach: that the network is uncongested as long as end-to-end losses are repaired "quickly enough". For this reason, RCRT uses the "time to recover loss*"* as a congestion indicator.

According to Figure 5, the out-Of-Order-List holds the packets that have been arrived since the first loss. Furthermore, this reflects the time that has passed to recover the first loss and we can use this indicator to decide whether the network is congested or not. RCRT wants the "time to recover" to be around one Round-Trip-Time (RTT) in order to avoid network collapse. Since the expected number of packets received during one RTT is ***ri*RTTi,*** if the out-Of-Order-List contains ***ri*RTTi*** then the network is not congested. Assuming the number of packets on the out-Of-Order-List is denoted by Li, RCRT uses the exponentially weighted moving average, denoted by Ci, to decide if the network is congestion or not. ***Ci=EWMA [Lnorm,i = Li/ri*RTTi]*** is a measure of network congestion derived from end-to-end reliability and uses thresholds such as an upper limit U=4 (network congested) and lower limit L=1 (network under-utilized) to ensure the best performance of the network. We use the **TTRMeasurement.class** where we calculate the exponentially weighted moving average (ewma()).

```
}
else if (expectedSegmentId < msg.getSegmentID()) {
    if (!myHandler.getOutOfOrderList().containsKey(msg.getSegmentID())) {
        myHandler.getOutOfOrderList().put(msg.getSegmentID(), Arrays.copyOfRange(msg.getPacket(), 0, msg.getBytesToWrite()));
        bytesOutOfOrderRead += msg.getBytesToWrite();
    }
    if (!recovery) {
        recovery = true;
    }
    else {
        continue;
    }

    Logger.log("Packet loss detected.");
            for (i = expectedSegmentId; i < msg.getSegmentID(); i++){
                missingPacketList.add(i);
                myHandler.addControlMsg(Configuration.MSG_CTRL_NACK+String.format("%08d", expectedSegmentId));
            expectedSegmentId++;
            }
            Logger.log("Missing packets: " + missingPacketList.toString());

    continue;
}
```

Figure 6: Java – Implementing the OutofOrderList to receive not expected packets

5.3. Rate Adaptation

RCRT uses AIMD (additive-increase/multiplicative-decrease) approach to confront the difficulties of rate allocation policies. 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. ***R(t)*** denotes the sum or current rates ***ri*** for all flows.

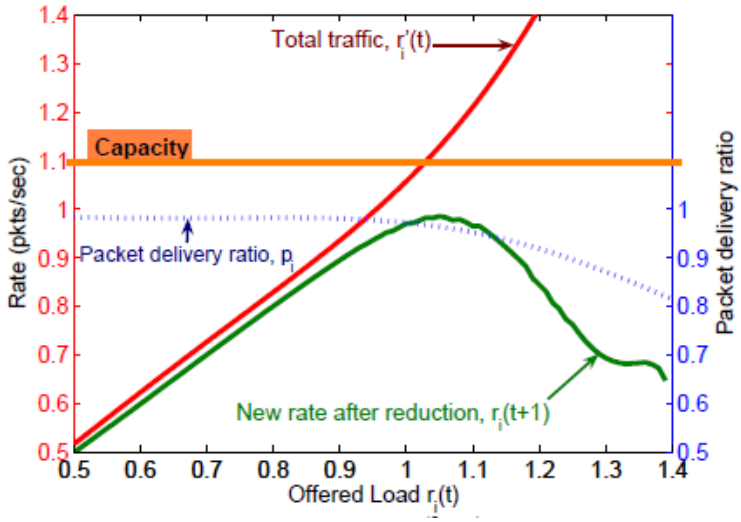

Figure 7: Packet delivery ratio, X-Axis is the offered load ri and Y-Axis represent both packet delivery ratio pi and the expected total traffic; r'i = [ri*(2-pi)]/pi

When the network is uncongested then RCRT increases additively:

R(t) : R(t+1)= R(t)+A, where A is a constant(eg 0.5 pkts/sec).

When the network is congested then RCRT decreases multiplicatively:

R(t+1)=M(t)*R(t), where M(t) is a time-dependent decreasing factor and defined by

M(t) = pi(t)/2-pi(t) where pi is the packet delivery ratio.

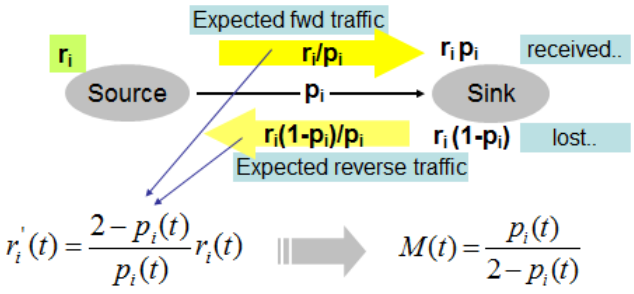

Figure 8: Explanation of decrease factor and packet delivery ratio

```
//Allocation Policy

//Fair - Equal Bandwidth for All - available bandwidth
if(Configuration.rateAllocationPolicy == 1){
    int equalShare = sensorsBW/SensorPool.getInstance().sizeSensors();
    System.out.println("equalshare"+equalShare);
    if(SensorPool.getInstance().getSensor(key).getCurrentRate()>0){
        SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d",equalShare));
    }
    SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d",equalShare));
    Logger.log("Applying a fair allocation policy for all sensors");
    Logger.log("Allocating asked rate "+equalShare+"KBps for sensor "+key);
}
//Demand-Proportional->In this case, Sensors get what they want!!!
else if (Configuration.rateAllocationPolicy == 2){
    float dratio = (float)sensorsBW/(float)sensorsBW;
    SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d", (int)(SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()*dratio)));
    Logger.log("Applying a demand-proportional allocation policy for all sensors");
    Logger.log("Allocating asked rate "+SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()+"KBps for sensor "+key);
}
//Demand-Limited - Rate Allocation ==3
else{

    int equalShare = sensorsBW/SensorPool.getInstance().sizeSensors();
    //for(int i=0;i<evKeySet.size();i++){
    //  sensorsCompareBandwidth.add(key,(int)SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced());
    //}
    if(SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()< equalShare){
        SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d",SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()));
        Logger.log("Applying a demand-limited allocation policy for all sensors");
        Logger.log("Allocating asked rate "+SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()+"KBps for sensor "+key);
    }
    else{
        sensorsBWOld = sensorsBW;//me hashmap??
        int remainingRate = maxUsedBW-sensorsBWOld;
            if(SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()< remainingRate){
                System.out.println("bw remins"+remainingRate);
                SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d",SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()));
                Logger.log("Applying a demand-limited allocation policy for all sensors");
                Logger.log("Allocating asked rate "+SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()+"KBps for sensor "+key);
            }
            else{
                SensorPool.getInstance().getSensor(key).addControlMsg(Configuration.MSG_CTRL_RATE+String.format("%08d",remainingRate));
                Logger.log("Applying a demand-limited allocation policy for all sensors");
                Logger.log("Allocating asked rate "+SensorPool.getInstance().getSensor(key).getSensorInfo().getRateAnnounced()+"KBps for sensor "+key);
            }
    }
}
```

Figure 9: The rate adaptation & Allocation mechanisms are utilized on the PreventionCongestion.class.

5.4. Rate Allocation

RCRT decouples the rate adaptation from rate allocation policy. Once the total rate ***R(t)*** is calculated by the rate adaptation mechanism, the role of RCRT's rate allocation component is to assign different rates ***ri(t)*** to each flow such that the individual flow rates sum up to $R(t)$, while keeping with the rate allocation policy ***Pj.*** This is crucial on sensor networks because there is no possible way to do an allocation a priori. RCRT contains three different policies:

Demand-proportional.

Each flow requests a rate to transmit data packets. This is called *demand* ***di***. This policy allocates rate ***ri*** to each node ***i*** proportional to its demand ***di*** such that the fraction (***ri(t)/di***) is the same for all sensors.

Demand-limited.

In this policy, the total rate of flows $R(t)$ is divided among all the flows equally, except that no flow gets a higher rate than the equal share rate. If a sensor ask for a lower than the equal share, it gets what it wants.

Fair.

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

The RCRT 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 *wi* (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 on the sink.

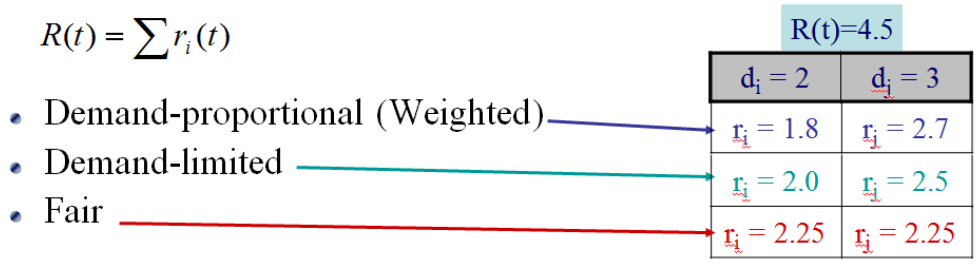

Figure 10: The rate adaptation & Allocation mechanisms are utilized on the PreventionCongestion.class.

6. Differences between RCRT and DISFER

While we are using the communication structure of DISFER, reusing the existing code for session control, we have to explain in a more detail manner the differences between the implementation of RCPT and the existing implementation of DISFER.

First of all, the end-to end reliability feature follows a different approach in DISFER. The sensor in DISFER does not immediately retransmit the missing packets. After the transmission is complete, the sensor retransmits the missing packets again to the sink. In contrast, in RCRT the missing packets are immediately retransmitted while the initial transmission gets to a halt. If there some are out of order of order packets, the sink temporarily stores them to a list while waiting for the missing packet to be retransmitted again. After the recovery completes, the out of order packets are passed to the application layer.

The congestion management of DISFER is based on RTT messages, while in RCRT is implemented by use of "Time-to-Recover, Ci". In both cases, each connection is being monitored using RTTi and Ci respectively. DISFER takes advantage of the separate channel/thread it has for control messages, while RCRT uses the "Out Of Order List" which resides in the sink. DISFER uses a percentage threshold of last few RTT samples/per sensor. If this percentage exceeds the value of a configurable threshold for four times repeatedly, then congestion exists. RCRT uses upper and lower threshold limits to indicate whether there is congestion or not. "Ci" indicates how many packets have been received since the first unrecovered loss but also how much time has passed since the first unrecovered loss. If the ideal average time taken to recover a loss is approximately one round trip time (RTT), the expected number of packets received during one RTT is *riRTTi*. If the out-of-order packet list length is *riRTTi*, then roughly one RTT has passed since the first unrecovered loss.

The rate adaptation and rate allocation algorithms of DISFER are composed by simple and fixed steps. The sink instructs the sensor to reduce its rate by 20%, if there is congestion on the network. In RCRT the AIMD approach from TCP is used to best utilize the available bandwidth of the network. As we explained on chapter 7, the rate adaptation algorithm adapts the total aggregate rate of all of the flows observed by the sink, rather than the rate of a single flow.

On the one hand, the DISFER protocol uses an equal share of available total rate if the requested bandwidth is higher than the available one; if the total request rate is below the available limit each sensor takes what it wants. On the other hand, RCPT uses 3 different policies which explained in section 8 by assigning rates to each flow such as individual flow rates sum up to R(t).

7. Appendix

7.1. Feedback Packets

Every feedback packet sent to a source *i* contains the assigned rate *ri*, a list of *NACK*ed sequence numbers, a cumulative ACK, and the ***RTTavg*** 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 send 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 ***RTOi***, defined as $\boldsymbol{RTTavg + 2 * RTTvar}$, where ***RTTavg*** is the average RTT and ***RTTvar*** 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.

7.2. 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 $\boldsymbol{T_i}$ when it sends a feedback packet to node *i*. Node *i* remembers the time $\boldsymbol{T'_i}$ at which it had received the feedback. When node *i* next sends a packet to the sink at Time $\boldsymbol{T''_i}$**,** it calculates the interval $\boldsymbol{T''_i - T'_i}$ and piggybacks this value in the packet. Upon reception of this packet at time $\boldsymbol{T'''_i}$, the sink can calculate the instantaneous RTT sample value $\boldsymbol{RTT_{inst,i}}$ by $\boldsymbol{(T'''_i - T_i) - (T''_i - T'_i)}$**.** RCRT uses an exponentially-weighted moving average of this value to get the estimated $\boldsymbol{RTT_{avg,i}}$ for node *i*. Earlier in this section, when we have referred to $\boldsymbol{RTT_i}$, we have meant $\boldsymbol{RTT_{avg,i}}$**.**

8. References

[1] TinyOS. http://www.tinyos.net/

[2] RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks (In *Proc. of ACM SenSys, 2007*), Jeongyeup Paek, Ramesh Govindan

[3] Sink Controlled Reliable Transport for Disaster Recovery (In *Proc. PETRA Conference, 2014*), Charilaos Stais, George Xylomenos, Giannis F. Marias.

[4] Congestion avoidance and control (In *Proc. ACM SIGCOMM Conference*, 1988), V. Jacobson.

[5] A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer (*ACM/IEEE Transactions on Networking*, 8(2):158–181, 1990), K. K. Ramakrishnan and R. Jain.