Error and Congestion Control for Wireless Sensor Networks

Charilaos Stais and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Athens 10434, Greece
stais@aueb.gr, xgeorge@aueb.gr

***Abstract*—In the Wireless Sensors Network (WSN) field, a wide variety of sensors produce a heterogeneous traffic mix, targeting diverse applications with different reliability requirements. We focus on emergency response scenarios, where a mobile rescuer moves through a, possibly disconnected, network, trying to talk to diverse sensors. We assume two types of sensors, event sensors triggered by an event and periodic sensors activated at predefined time intervals, as well as two types of transmission, either using the highest bit rate available or using predefined bit-rates. Our Reliable Transport protocol for SEnsor Networks with MObile Sinks (RT-SENMOS), takes into account all these parameters and tries to provide the best possible user experience under the current circumstances of the network, using a sink-driven approach where an application-specific sink is combined with generic sensors. RT-SENMOS was implemented and tested over a real network with emulated losses and compared against Rate-Controlled Reliable Transport (RCRT), a well-known sink-driven protocol. The results show that RT-SENMOS fully exploits the available bandwidth in all cases, while RCRT only manages to exploit 60% to 90% of it. Furthermore, RT-SENMOS adapts much faster to prevailing network conditions, while its protocol overhead, in terms of control messages exchanged, is much lower than that of RCRT.**

***Index Terms*—WSN, Transport, Error Control, Congestion Control**

I. Introduction

Among the wide range of applications of Wireless Sensor Network (WSN), one of the most demanding ones is emergency response in disaster areas, both because the WSN may have become partitioned due to sensor failures, and because of the need to transport data quickly and accurately to the emergency response team. A human or a robot entering a disaster area to provide help can expect the network to be connected or all the sensors to be available, therefore emergency response applications need a transport protocol that can connect to sensors quickly and transfer as much data as possible reliably, without creating congestion around the rescuer where all the data streams are converging. This means that a transport layer protocol for emergency response applications must not only handle the unpredictability of regular WSNs, it must also constantly adapt to the shifting congestion levels and network connectivity created by the rescuer's mobility.

As part of the the DIstributed Sensor systems For Emergency Response (DISFER) project, we created and implemented the *Reliable Transport protocol for SEnsor Networks with MObile Sinks* (RT-SENMOS) [1]. To better handle emergency response applications, RT-SENMOS moves *all* responsibility for error and congestion control to the sink, in our case represented by a mobile rescuer; that is, the sink instructs the sensors what and how much to send, with the sensors implementing a generic set of mechanisms. The rationale behind this design decision is to allow the application to fully control the transport session depending on its requirements, for example, allocating higher transport rates to more important sensors and adjusting the reliability level depending on the sensor data. As a result, a set of generic sensors normally used by a totally unrelated application, e.g., security, can be leveraged for an emergency response application targeted to the specific disaster scenario. The mobile rescuer simply loads to the sink an application appropriate for the disaster scenario at hand (e.g., fire, flood, earthquake, etc.) and enters the disaster area, using any already installed sensors that remain operational. Finally, unlike most transport protocols which require support at the kernel level, RT-SENMOS runs entirely in user space over UDP/IP as part of the application executing at the sink.

In this paper we present the design of RT-SENMOS, including a performance evaluation of RT-SENMOS against the Rate-Controlled Reliable Transport (RCRT) [2] protocol, another sink-controlled approach, using actual implementations of both protocols running on a real network with emulated losses, extending the preliminary evaluation of our previous work [1]. While RCRT is very similar in philosophy to RT-SENMOS, it is not as appropriate for emergency response applications for four reasons. First, RCRT takes a long time to detect congestion, as it monitors loss recovery time; RT-SENMOS uses timeouts to detect congestion faster, which is critical for mobile rescuers. Second, RCRT responds too drastically to congestion (similarly to TCP); RT-SENMOS uses fixed adaptation steps to avoid dropping the transmission rate to zero during rescue missions. Third, while RCRT offers different rate allocation policies, they apply to all sensors; RT-SENMOS can apply different rate allocation policies to each sensor type, allowing the application to prioritize the most important sensors. Finally, while RCRT sends original and retransmitted packets together, RT-SENMOS can either recover from errors with immediate retransmissions or in recovery rounds, allowing the application to prioritize the

recovery of the most important data.

The second version of RT-SENMOS described in this paper implements two different congestion control policies, covering both sensors that can transmit at any available rate and sensors that can only transmit at specific rates (e.g., video cameras), and two different error control methods, either immediate retransmissions or recovery rounds, superseding the simpler first version that only implemented congestion control for sensors transmitting at any available rate and recovery in rounds [3], [4]. Essentially, the first version of RT-SENMOS only offers a subset of the functionality of the second version. We apply both RT-SENMOS and RCRT to an emergency response scenario, where different types of sensors connect with a mobile rescuer. The results show that RT-SENMOS fully exploits the available bandwidth in all cases, while RCRT only manages to exploit 60% to 90% of it, it adapts much faster to prevailing network conditions and requires far less control messages to operate.

The structure of the remainder of this paper is as follows. In Section II we present past work on WSN transport protocols and place RT-SENMOS in this context. In Section III we present our assumptions and motivate our design choices. In Section IV we describe in detail the mechanisms and policies of RT-SENMOS. Section V contains our performance evaluation of RT-SENMOS against RCRT in an emergency response scenario. We conclude in Section VI.

II. Background and Related Work

A. Types of WSN transport protocols

A characteristic aspect of WSNs is that their typical use involves a large number of sensor nodes communicating with a single sink, either directly or via other nodes (sensors or not). This many-to-one communication model requires different assumptions than those made for one-to-one transport protocols, such as TCP. In this section we discuss existing work on reliable transport for WSNs, with a focus on their suitability for emergency response scenarios, where the WSN may only be partially connected, as some sensors may have failed. A rescuer may thus have to roam around the disaster area to receive information from the surviving sensors. As speed is essential in emergency response, transport protocols for WSNs in such scenarios must be able to quickly connect to sensors, get as much information as possible, and adapt to shifting network conditions as the rescuer roams.

WSN transport protocols can be categorized using two orthogonal axes [2]. First, in the reliability axis, a protocol may provide either a reliable or an unreliable service. Second, in the congestion control axis, a protocol may provide no congestion control, distributed congestion control or centralized congestion control. Reliability can be further implemented hop-by-hop with retransmissions, as in RMST [5], end-to-end with retransmissions, as in RCRT [2] and STCP [6], or end-to-end using forward error control, as in ReInForM [7]. While hop-by-hop reliability is useful for WSNs with losses due to bad link conditions, most WSNs can retransmit lost packets at the link layer, if needed [5]. The use of forward reliability between acknowledgments requires a well-connected WSN, which is unlikely in cases of emergency response applications. Considering the fact that end-to-end retransmissions do not require any co-operation from intermediate nodes, we will focus on this kind of recovery scheme.

We can also distinguish WSN transport protocols based on who is responsible for congestion control. In sensor-driven protocols, each sensor sending data is responsible for congestion detection and error correction in its own data stream. In this approach, the sink simply receives data and reports packet losses (through ACKs or NAKs). The advantage of this technique is that the sink is very simple. However, this means that the sensors must be more complex as they must make all the decisions. The other option is to make the sink drive congestion control, using its awareness of all simultaneous transport sessions to co-ordinate the sensors. In this case, the sink is complex and the sensors are simple; in addition to its cost benefits, this choice can enable different applications to be supported by suitably modifying the sink. This is of particular interest in emergency response scenarios, where sensors that may have been intended for a different application (e.g., security) may be reused to assist rescuers. For this reason, we will focus on sink-driven schemes.

An alternative to the end-to-end handling of error and congestion control is hierarchical aggregation. For example, ESRT [8] chooses some nodes and assigns them the responsibility to manage those events, by collecting data from nearby sensors, merging them and delivering them to the sink. This approach comes with the problem of selecting merge nodes, which implies network topology discovery. Also, a hierarchical approach may cause delays in cases of emergency where we need quick responses.

Another way to classify transport protocols is based on how they detect congestion. The main methods are observing packet losses, queue lengths and packet delays, either in absolute terms or in relative terms (i.e., how these metrics evolve over time). Packet loss measurements require an ACK or NAK-based mechanism for feedback, while delay measurements require taking Round-Trip Time (RTT) samples; queue length measurement does not require feedback. All these techniques are in principle suitable for emergency response applications.

B. Reliable transport protocols for WSNs

The Priority-based Congestion Control Protocol (PCCP) [9] detects congestion using a metric derived from the ratio of packet inter-arrival time to packet service time. The purpose of this metric is to improve QoS via better link utilization with respect to packet losses and delays. The Adaptive Rate Control (ARC) [10] protocol does not enforce a specific method of congestion detection and management; each node acts alone and tries to adapt its transmission rate. This approach does not burden the network with control overhead, but speed and efficiency cannot be guaranteed. Siphon [11] reacts to congestion by redirecting traffic to other nodes, which act as virtual sinks. The goal is to unload congested links and perform load balancing through multipath transmission. This approach assumes a multi-hop and dense network, where we can easily find alternative paths towards the sink.

Multipath transmission and load balancing are combined in [12], aiming to proactively handle congestion via a primary and a secondary path, if available, for load balancing. The mechanism monitors the sensor's buffers for congestion detection. In the same direction, CODA [13] takes advantage of buffer monitoring to detect congestion. The sink examines the usage of nearby buffers based on sensor reports and uses this information to detect congestion. Once congestion appears, the sink starts sending signals and all other nodes push them via a backpressure mechanism towards the source. In a similar way, Fusion [14] relies on the queue lengths of intermediate nodes for congestion detection and uses hop-by-hop rate adjustment. Buffer usage is normally a good indicator of network capacity. However, when collisions are excessive, after several unsuccessful MAC-level retransmissions packets are dropped, therefore a decrease in buffer occupancy may not indicate the absence of congestion.

In [15] the authors combine fairness and congestion control, assuming that fairness is achieved via the allocation of equal rates to each node. This solution is not suitable for WSNs with different types of sensors, where each type demands a different rate. The use of an equal rate allocation policy may not exploit all the available bandwidth, leaving some nodes with less bandwidth than what they asked for, while wasting bandwidth when for nodes that do not need it.

Rate-Controlled Reliable Transport (RCRT) [2] adopts a centralized approach where the sink is responsible for network monitoring and congestion management, leaving sensors without any functionality related to reliable transfer (except recovery) as in STCP [6]. RCRT employs a negative acknowledgment mechanism, where only losses are reported by the sink to the sensors. RCRT can employ three different rate allocation policies to assign the available transmission capacity to the sensors. In the *fair* policy, the same rate is allocated to all sensors, regardless of their requirements. In the *demand proportional* policy, each sensor is assigned the same fraction of its desired rate as all other sensors. In the *demand limited* policy the same rate is allocated to all sensors, except that sensors that asked for a lower rate are only assigned that rate, leaving more bandwidth for the others. RCRT relies on the time needed to recover from a packet loss to detect congestion and adapt the overall transmission rate to be assigned to the sensors. If this time is too high, the network is congested and the sink reduces the assigned rates. If it is too low, then the network is underutilized and the sink increases the assigned rates. If it is in between, the network is stable and rates do not change. RCRT is a good candidate for emergency response applications as the sink is in control of the network, meaning that sensors are simple, all control loops are end-to-end, meaning there is no reliance on intermediate nodes, and the protocol is relatively simple to operate, meaning that it is easy to re-establish connectivity when the sink moves. However, the only distinction possible between sensors is their required transmission rate.

III. Assumptions and Design Rationale

The only assumption made by RT-SENMOS about the network is that it is a multi-hop WSN using a broadcast-based wireless communication technology, which could be WiFi, Bluetooth or ZigBee. Some of the nodes in the network (possibly, all of them) host sensors, which try to transmit their data to a pre-defined network address, the sink. The broadcast communication technology implies that the sink is a natural congestion point, as all data transmissions from the sensors converge there and have to contend for the same bandwidth. Since we target emergency response scenarios, we assume that the sink may be a human or robotic rescuer moving around the disaster area. This implies that the WSN from the viewpoint of the sink may be constantly in flux. We assume that all sensor nodes are pre-programmed with a network address for the sink, but they are otherwise generic, in the sense that they just try to send their data to the sink using a set of generic mechanisms. The application logic is embedded in the sink, which exploits the generic RT-SENMOS mechanisms to achieve its specific goals. This allows the WSN to support arbitrary applications by simply having an appropriate application at the sink assume the pre-defined sink address.

In general, the WSN may be partitioned due to sensor and node failures, therefore as the sink moves it may connect with different partitions of the network from different vantage points. We assume an underlying WSN routing protocol that dynamically connects the mobile sink with any reachable sensors [16], understanding that connectivity with a sensor may fail at any time. Due to our focus on emergency response scenarios, we do not expect the network topology to be complicated; instead, we expect that sensors will be only a few (probably only one) hop away from the sink, with the sink having to move across the area to communicate with additional sensors. For this reason, rather than rely on backpressure mechanisms for congestion management, which would be undermined by the constant sink movement, we instead rely on end-to-end congestion and error control, as it only requires co-operation between the sink and each individual sensor.

To allow RT-SENMOS to be used for a variety of applications, we need to accomodate diverse sensor types by treating them appropriately. Sensors can choose among two types of error recovery behavior, which have to be partially implemented at the sensor; each sensor will generally only implement one behavior. Bandwidth allocation however is controlled fully by the sink, without any support from the sensors, therefore each application can use its own algorithms. We have implemented a two phase bandwidth allocation policy, which firsts splits the available bandwidth among different sensor types, and then dynamically allocates a portion of that bandwidth to each individual sensor of that type. Dynamic allocation takes into account the sensor type, its desired transmission rate(s) and the behavior of a network where congestion is naturally concentrated around the sink.

RT-SENMOS is implemented at the application-layer over UDP/IP, allowing it to be embedded in practically any device, without requiring kernel modifications or superuser privileges. Our implementation is written in Java, therefore it can be used on all kinds of devices, including Android smartphones and tablets. While most transport protocols are implemented as libraries, the RT-SENMOS sink implementation was designed to be embedded within an application, thus allowing

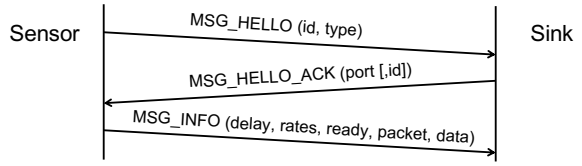


Fig. 1: Connection establishment and sensor information exchange.

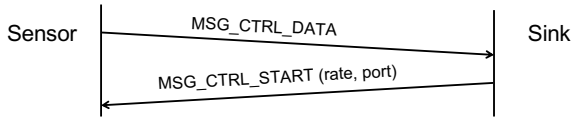


Fig. 2: Start of a new data exchange.

it to directly control all protocol parameters depending on its needs, following the *Application Layer Framing* (ALF) model [17]. In addition to the rate allocation policies between and within each type, the application can also control the level of reliability desired. The sensors are, again, oblivious to all these policies. This allows different applications, customized for different emergency response scenarios (e.g., fire, flood, earthquake, etc.), to exploit any sensors already available in a disaster area; since all the application logic is in the sink, a sensor supporting RT-SENMOS can be exploited by any RT-SENMOS compatible application.

IV. Protocol Description

This section describes the operation of the second version of RT-SENMOS[1]. From the viewpoint of a sensor, the protocol operates in three phases. First, a sensor makes periodic attempts to connect to the sink; when the sink responds, the sensor sends it characteristics, and the sink responds with an initial rate allocation. Then, a number of data transmission rounds take place, with retransmissions either alongside the original transmissions or after them, in recovery rounds. During this pase, the sink may update the rate allocation of a sensor at any time with a rate update message. Finally, either side can drop the connection or detect (via timeouts) a connection failure. A more detailed description is provided in the following subsections.

A. Connection establishment

In RT-SENMOS, control signaling takes place out-of-band, using separate data and control channels. All connections are initiated by the sensors which send communication requests to the pre-defined sink address using a fixed UDP port, essentially representing a common control channel. The sink then allocates a UDP port to the sensor for signaling and a separate UDP port for data transport. In a multithreaded environment, the sink can use one thread to receive connection requests from new sensors, one thread per sensor for signaling with known sensors and additional per-sensor threads for data transmissions. The sensors can in turn uss one thread for signaling and another thread for data. This scheme allows control messages to be sent without delay, which is especially useful during congestion period.

The sink constantly listens to the common control channel for connection requests. Sensors can either periodically try to connect to the sink or, if the routing protocol offers this option, wait until the sink becomes reachable. In both cases, the sensor sends an `MSG_HELLO` message to the common control channel, as shown in Figure 1, which includes the pre-configured identifier of the sensor and its type (for example, event or periodic). When the sink receives an `MSG_HELLO` message, it respondes with a `MSG_HELLO_ACK` message which includes the UDP port for the signaling channel dedicated to that sensor; if the sensor left the identifier field empty in its `MSG_HELLO` message, the `MSG_HELLO_ACK` message also includes an identifier allocated by the sink.

When the sensor receives an `MSG_HELLO_ACK` message, it prepares an `MSG_INFO` message, which is sent, as all subsequent control messages, to the signaling channel dedicated to that sensor. This message includes the delay between receiving the `MSG_HELLO_ACK` message from the sink and sending the `MSG_INFO` message, the set of data transmission rates supported by the sensor (possibly, only one), an indication of whether the sensor is ready to send data immediately, in which case the message serves as a request to send, as well as the size of the data packets supported by the sensor and the total size of the data that the sensor will send. The sink, upon reception of the `MSG_INFO` message, can calculate an initial Round-Trip Time (RTT) estimate for that sensor by subtracting the delay given in the `MSG_INFO` message, which represents processing time at the sensor, from the time period between sending the `MSG_HELLO_ACK` message and receiving the `MSG_INFO` message, which is measured at the sink side. The sink is now aware of the sensor's requirements and can proceed with rate allocation and data transmission.

B. Data exchange and idle

When a sensor sends its initial `MSG_INFO` message to the sink, it may indicate that it is ready to send data, in which case the sink can proceed with allocating a rate to the sensor and giving it the go-ahead to transmit. Otherwise, the connection moves to an idle state until the sensor sends a `MSG_CTRL_DATA` message, indicating that it is now ready to send data, as shown in Figure 2. The same procedure is repeated after every transmission, that is, the connection becomes idle until the sensor sends a new `MSG_CTRL_DATA` message. This message does not include any parameters, since all relevant parameters have been passed to the sink via the `MSG_INFO` message.

The sink directs the sensor to start transmitting by sending it an `MSG_CTRL_START` message which indicates the data rate that the sensor should use as well as the UDP port to be used for the data transmission, which represents a sensor-specific data channel. The rate allocated to the sink is set as explained in Section IV-E. The sensor then starts sending data (and, possibly, retransmissions) using the data channel,

[1]Older versions of the protocol used a simpler congestion control scheme and a single error control method; see [3], [4] for details.

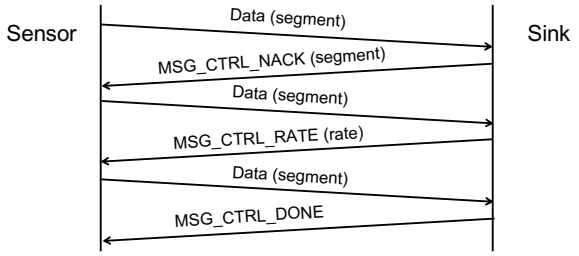


Fig. 3: Data exchange.

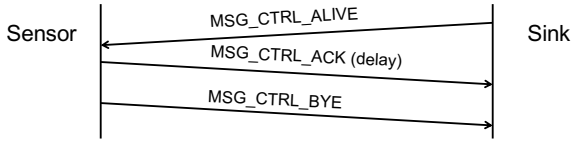


Fig. 4: Connection control and release.

while the sink sends control messages, such as Negative Acknowledgments (NAKs) and rate updates, over the signaling channel. The sensor keeps sending data packets with the size indicated in its `MSG_INFO` message, until the total data, again as indicated in its `MSG_INFO` message, are exhausted. The data packet header holds a segment number used to sequentially number the packets; retransmissions use the same number as the original packet.

When a lost data packet is detected, either due to a missing segment number, or due to a long delay between packet arrivals, the sink returns a `MSG_CTRL_NAK` to the sensor over the signaling channel, as shown in Figure 3. The error recovery measures taken by the sensor depend on its type, as explained in Section IV-D. Once the data transmission, including any recovery messages, is done, the sink indicates that the transfer completed successfully by returning a `MSG_CTRL_DONE` message over the signaling channel. Note that the sink knows the total size of the data transfer from the `MSG_INFO` message, therefore it can detect missing packets at the end. Both endpoints the move back to the idle state, until the sensor is again ready to transmit. The sink may send a `MSG_CTRL_RATE` message to the sensor at any time indicating a modified rate allocation, as explained in Section IV-E.

C. Connection control and release

Due to sink mobility, the connection between the sink and the sensor may be broken at any time, without any indication to the endpoints. To monitor the state of the connection, the sink periodically sends `MSG_CTRL_ALIVE` messages to the sensor, which are acknowledged by `MSG_CTRL_ACK` messages from the sensor, as shown in Figure 4. The `MSG_CTRL_ACK` includes the processing delay at the sensor, thus allowing the sink to not only confirm that the connection is alive, but also to make a fresh estimate of the RTT of the connection to that sensor, by subtracting this delay from the time taken between sending the `MSG_CTRL_ALIVE` message and receiving the `MSG_CTRL_ACK` message.

On the other hand, if any of the endpoints wants to terminate the connection voluntarily, it can send a `MSG_CTRL_BYE` message to the other side. This message is not acknowledged, leading to an immediate drop of the connection at both sides. If it is lost, the other side will eventually timeout, either waiting for an `MSG_CTRL_ALIVE` message to arrive (sensor) or waiting for an `MSG_CTRL_ACK` message to be returned (sink).

D. Error recovery

In RT-SENMOS the sink generates NAKs for missing data packets and the sensors retransmit those packets; segment numbers are used to detect lost data packets, while RTT-based timeouts are used to detect lost control packets or very delayed data packets. The exact error recovery policy can be different for each type of sensor, but as part of the retransmission mechanism is implemented at the sensors, we have designed two different recovery schemes to cover as many applications as possible, while still offering to the sink the opportunity to adjust the reliability level depending on the application, without involving the sensor.

In our test scenarios we have two types of sensors. A *periodic sensor* periodically collects data, for example, the current temperature or a still picture from a security camera. An *event sensor* is triggered by a specific event, for example, motion detection or a fire alarm, to send some data, for example, a short video from a security camera. Since event sensors are triggered by an individual event, we assume that their data transmissions are delay sensitive as we might need to know their readings right after the event. On the other hand, periodic sensors constantly transmit new data, therefore we assume that they are not as delay sensitive, since we just need to know what their latest available reading is.

For the non delay-sensitive *periodic* sensors, we retransmit missing packets in *recovery rounds*. First, the sensor transmits all the original data packets and the sink returns NAKs for any missing ones. Then, all data packets for which NAKs were received are retransmitted. If NAKs are received for the retransmissions, we retransmit again the NAKed packets, repeating this procedure until all packets are received, as in RMTPSI [18]. The sink can stop the recovery process whenever it wants, for example, when enough packets are received to reconstruct the content, by sending a `MSG_CTRL_DONE` message. The goal of this policy is to provide an overview of all the data quickly, and then let the application decide how many missing packets it wants to recover.

For the delay-sensitive *event* sensors, packets are immediately retransmitted by the sensors when a NAK for them is received, as in RCRT [2]. The sink can choose which packets to NAK, depending on their value to the application. For example, when transmitting an MPEG-encoded video, packets containing I-frames are more valuable than those containing P-frames, since I-frames are needed to decode P-frames. In our evaluation which uses video cameras as event sensors, the sink monitors the recovery time of the past n retransmissions to determine whether a retransmitted packet will arrive soon enough to be played out, based on the current position of the

video player; if not, the packet is not NAKed, since it would waste transmission resources.

E. Bandwidth management

The bandwidth management algorithm relies on two observations about our environment: first, in a broadcast-based WSN congestion is naturally concentrated around the sink and, second, the algorithm must adapt very quickly due to the mobiilty of the sink. As a result, while TCP uses an Additive Increase - Multiplicative Decrease (AIMD) [19] algorithm to probe the (unknown) bandwidth on the end-to-end path, leading to wild rate fluctuations, RT-SENMOS starts from the (known) bandwidth at the sink, which is determined at the MAC level and adds *fixed* adaptation steps.

RT-SENMOS first assigns a fraction of the available bandwidth to each type of sensor, depending on the priorities of each application. Then, it allocates rates to individual sensors, possibly using a different algorithm per sensor type. Note again that the sensors are not aware of the actual scheme used. Initial rate allocation takes place whenever the sensor mix changes, that is, when a new sensor connects to the sink or when an existing one gets disconnected. Between these events, the rates are modulated by a congestion management scheme that measures the loss rate from each sensor in order to adapt its transmission rate.

In the first version of RT-SENMOS sensors indicated a single desired rate during connection establishment [4]; in the second version we also allow indicating a set of target rates, possibly with a single member [1]. In our experimental evaluation, periodic sensors ask for a single rate, while event sensors ask for a set of rates. The algorithm interprets the single rate as the minimum needed to offer acceptable service, so the rate allocation algorithm tries to match or exceed it. On the other hand, with multiple rates the algorithm assumes that the sensor can use any of these rates, prefering the highest one, meaning that the rate allocation algorithm should exactly match one of these rates, otherwise bandwidth may remain unused. We next explain how the rates are initially allocated and then dynamically adapted.

1) Initial Rate Allocation: Every time the sensor mix changes, that is, sensors are connected or disconnected, the initial rate allocation algorithm assesses whether it needs to make global adjustments, separately for each type of sensor. For periodic sensors, we first calculate the sum of the rates requested (not assigned) by all sensors of that type. If this is below the available bandwidth for this sensor type, new sensors and existing sensors which were rate-limited will get their requested rate, while all other existing sensors will keep their assigned rate. On the other hand, it the aggregate requested bandwidth is more than what is available, the available rate is shared *equally* among all sensors of that type. Note that if the event that triggered the algorithm was a sensor leaving the mix, the algorithm first checks if the current rate of a sensor is higher than its requested rate. If so, there is no need to change it, since the overall available bandwidth must have increased. The detailed procedure is given in Figure 5.

On the other hand, for event sensors which only support specific rates, when the bandwidth is not enough to satisfy all requests, the equal share rates initially assigned to each sensor must be adjusted to the highest supported rate that is less than or equal to their initial allocation. As a result, some of the initially allocated bandwidth may remain unused; we will allocate this bandwidth to event sensors whose allocated bandwidth is smaller than their desired bandwidth and are not in a congested state. The rate allocation scheme for event sensors is elaborated upon in Figure 6; the algorithm for assigning the remainder is explained below, as part of the dynamic rate adaptation algorithm.

```
procedure RATE_ALLOCATION_PERIODIC
    periodic_total_bw := calculate_total_req_BW(periodic_sensors_list)
    periodic_BW_share := get_total_BW() * (1 - config.get_event_share())
    if periodic_total_bw < periodic_BW_share then
        for s in periodic_sensors_list do
            if NOT(left_event AND s.current_rate > equal_share) then
                assign_rate(s.req_rate)
            end if
        end for
    else
        equal_share := periodic_total_bw/num_of_periodic_senso
        for s in periodic_sensors_list do
            if NOT(left_event AND s.current_rate > equal_share) then
                assign_rate(equal_share)
            end if
        end for
    end if
end procedure
```

Fig. 5: Initial rate allocation for periodic sensors.

After all bandwidth has been allocated, new sensors are notified of their assigned rate in the start transmission message, while existing sensors whose allocation has changed are notified by a rate update message (see Section IV-B).

2) Dynamic Rate Adaptation: While the sensor mix remains static, the system periodically detects congestion and adapts the assigned rates to prevailing conditions as shown in Figure 7. The congestion detection scheme of RT-SENMOS assumes that the congestion induced losses in each connection are more similar to those occurring in a network with Random Early Detection (RED) [20] rather than in a network with drop-tail gateways. The rationale is that the large number of sensors involved and the convergence of all transmissions around the sink will randomly distribute losses among the connections, with a loss probability proportional to the level of congestion. Furthermore, we assume that each sensor type expects a minimum reliability level, expressed by a loss rate threshold $loss\_limit$; this accounts for wireless losses that do not indicate congestion. The dynamic rate adaptation scheme monitors the current loss rate, $current\_loss$, for each sensor, and uses the gap between $loss\_limit$ and $current\_loss$ to make its decisions.

```
procedure RATE_ALLOCATION_EVENT
    event_total_bw := calculate_total_req_BW(event_sensors_list)
    event_BW_share := get_total_BW() * config.get_event_share()
    unused_BW := 0
    if event_total_bw < event_BW_share then
        for s in event_sensors_list do
            if NOT(left_event AND s.current_rate > s.req_rate) then
                assign_rate(s.req_rate)
            end if
        end for
        unused_BW := event_BW_share - event_total_bw
    else
        equal_share := event_total_bw/num_of_event_sensors
        for s in event_sensors_list do
            if equal_share NOT in list of supported rates by sensor then
                new_rate := find_suitable_bitrate(equal_share)
                unused_BW+ := equal_share - new_rate
            else
                new_rate := equal_share
            end if
            if NOT(left_event AND s.current_rate > new_rate) then
                assign_rate(new_rate)
            end if
        end for
    end if
    assign_unused_BW(event_sensors_list, unused_BW)
end procedure
```

Fig. 6: Initial rate allocation for event sensors.

```
procedure CONGESTION_HANDLER
    for s in sort_loss_rate_decrease(event_sensors_list) do
        if s.current_loss > s.loss_limit then
            new_rate := s.current_rate * (1 - s.current_loss)
            assign_rate(find_suitable_bitrate(new_rate))
        end if
    end for
    for s in sort_loss_rate_decrease(periodic_sensors_list) do
        if s.current_loss > s.loss_limit then
            new_rate := s.current_rate * (1 - s.current_loss)
            assign_rate(new_rate)
        end if
    end for
    Wait for changes to take effect
    for s in sort_loss_rate_increase(event_sensors_list) do
        if s.current_loss < s.loss_limit then
            new_rate := s.current_rate * (1 + s.loss_limit - s.current_loss)
            assign_rate(find_suitable_bitrate(new_rate))
        end if
    end for
    for s in sort_loss_rate_increase(periodic_sensors_list) do
        if s.current_loss < s.loss_limit then
            new_rate := s.current_rate * (1 + s.loss_limit - s.current_loss)
            assign_rate(new_rate)
        end if
    end for
end procedure
```

Fig. 7: Dynamic rate adaptation.

Specifically, when $loss\_limit < current\_loss$, the sink decreases each rate proportionally to the current loss rate: $newRate = currentRate * (1 - current\_loss)$. On the other hand, when $loss\_limit > current\_loss$, the sink increases each rate proportionally to the difference between the limit and the current loss rate: $newRate = currentRate * (1 + loss\_limit - current\_loss)$. As an example, if $loss\_limit = 3\%$, when $current\_loss = 1\%$ the rate will be increased by 2%, while when $current\_loss = 5\%$ the rate will be decreased by 5%; note the slightly higher aggressiveness when rates are reduced. The event sensor rates are then modified as explained in the previous section, that is, we adjust them to the highest feasible desired rate.

Dynamic rate adaptation is performed in two steps, as explained in Figure 7. In the first step, we examine all sensors that may be congested, starting from the sensors with the highest loss rate and proceeding towards those with the lowest loss rate. Any sensors found to be in a congested state, i.e., exhibiting a loss rate higher than the threshold, are assigned reduced rates. After that, we wait for the network to settle, and then proceed in the second step, where we examine all sensors that may not be congested, from lowest to highest loss rate. Any sensors found to be in an uncongested state, are assigned increased rates.

Note that the second part of Figure 7, where rates are increased, is also used as the last step in the initial rate allocation algorithms for event sensors (see Figure 6), in order to exploit any leftover bandwidth from the event sensor pool: we cycle through the event sensors and only assign the leftover rate to a sensor if it allows the sensor to upgrade its rate to the next possible one, e.g., the next highest video quality; whatever is left, is assigned to the periodic sensor pool.

V. PERFORMANCE EVALUATION

A. Experimental setup

To evaluate the performance of RT-SENMOS against RCRT, we employed three different scenarios. The main concept in all scenarios is the emulation of a disaster area, with different sensor setups in each scenario. In the *event sensor* and *periodic*

TABLE I: Experimental parameters

Parameter	Value
Content size for periodic sensors (MB)	8
Bit rate for content sensors (KBps)	82
Content duration for event sensors (sec)	30
Bit rates for event sensors (KBps)	50, 87, 187 and 312
Bandwidth available at the sink (MBps)	2
Chunk size (bytes)	512
Target loss rate	2% or 5%
Loss rate threshold	2% or 5%
Periodic sensor share (mixed scenario)	10%, 30% and 50%

sensor scenarios we have 14 sensors of the corresponding type, while in the *mixed sensor* scenario we have 14 event and 14 periodic sensors. In all scenarios, an additional node acts as the sink, connected over a single hop to all sensors in range, using a shared broadcast WiFi link. We emulated the movement of the sink in the disaster area by having sensors get in range (and get connected to) the sink with a specific pattern in each scenario, while congestion induced losses towards the sink were emulated using a programmable loss injection module.

While losses were independent, the loss rate emulated was proportional to the current bandwidth allocation, since we assumed that the congestion loss model was RED-like. Specifically, to calculate the loss rate, we multiplied the fraction of the total bandwidth that was allocated to sensors with a target loss rate, which was either 2% or 5%; thus, the effective loss rate ranged from zero (no bandwidth allocated) to the target loss rate (all bandwidth allocated). The loss rate threshold for each sensor with RT-SENMOS, below which we assumed there was no congestion (the acceptable loss rate), was also either 2% or 5%.

We used our own implementation of the second version of RT-SENMOS written in standard Java[2]. Using the same Java code base and messaging scheme, we also implemented the full functionality of RCRT, including all of its rate allocation policies; the parameters were set as in the paper proposing RCRT [2]. The results provided below for RCRT are from the fair and demand-proportional policies, where sensors get either the same rate or the same proportion of their desired rate, respectively. For event sensors, since RCRT does not support the concept of multiple bit rates, we used the highest desired rate as the target rate. Since the rate-limited policy of RCRT interprets the desired rate as the highest rate desirable, it did not make sense to compare it with RT-SENMOS which tries to exceed this rate with periodic sensors.

As periodic sensors we used still cameras that periodically transmitted 8 MB snapshots at a minimum desired rate of 82 KBps. As event sensors we used video cameras that, when triggered, sent 30 sec of live video at 50, 87, 187 or 312 KBps; the actual rate used depended on their allocated rate. The bandwidth available at the sink was 2 MBps (or, 16 Mbps). The share of the total bandwidth allocated to periodic sensors was 10%, 30% and 50%. In all cases, data was transmitted in 512-byte chunks. The experimental parameters are listed in Table I.

[2]An Android Java version is also available, which interoperates with the standard Java version.

The *event sensor* scenario simulates an emergency response incident where a rescuer enters the disaster area and tries to get a short video from each camera. During the first 2 sec of the experiment the event sensors gradually connect to the mobile rescuer, and then we gather data until all sensors have completed their transmissions. In the *periodic sensor* scenario we keep the same setup as in the *event sensor* scenario, but the periodic sensors transmit a single fixed-size screenshot each. Since in both scenarios each sensor only makes a single transmission, the main difference between the scenarios is in the way each type of sensor operates: event sensors can only use specific rates and recover from loses in parallel with data transmissions, while periodic sensors can use any assigned rate (ideally, higher than the target rate) and recover from losses in rounds. It should be noted that the first version of RT-SENMOS only supported this type of scenario, that is, sensors that can transmit at any available rate with error recovery in rounds.

Finally, in the *mixed sensor* scenario we assumed the existence of a large hall where all periodic sensors were installed, leading to a 100 m corridor in which event sensors were installed every 6.8 m. The rescuer moves from the large hall to the corridor, connecting to the event sensors as they get in range, while the periodic sensors are in range from the beginning. Again, we gather data until all transmissions finish. In all scenarios, there was no significant difference in experiment completion time between RT-SENMOS and the variants of RCRT, so the focus of the evaluation was on the behavior of the rate allocation schemes.

B. Experimental results

1) Event sensor scenario: We first discuss the results from the *event sensor* scenario, where sensors can only take advantage of specific data rates. In Figure 8 we show the mean bandwidth allocated and used per sensor, for loss rate thresholds of 2% and 5%, and a target loss rate of 2%. Note that since losses are tracked on a per sensor basis, even though the average loss rate peaks at 2%, individual connections can see much higher loss rates, thus triggering congestion control actions. Although in both cases RT-SENMOS quickly adapts as sensors enter and leave the network, the match between the assigned and used rates is much better when the loss rate threshold is set to 5%, higher than the target loss.

In Figure 9a we compare RCRT[3] and RT-SENMOS with a 5% loss rate threshold, with a target loss rate of 2%. As in the previous figure, the allocated and used bandwidth in RT-SENMOS is almost the same, meaning that RT-SENMOS allocates rates which can actually be used by the sensors, while RCRT exhibits a remarkable difference between the allocated and used bandwidth, as it strives to allocate as much bandwidth as possible to each sensor, rather than the actual rate that the sensor can use, thus wasting the available bandwidth. Note also that RCRT slowly increases the allocated bandwidth, since there is no congestion (as some of the allocated bandwidth

[3]We used the demand-proportional policy for this figure, but the fair policy has exactly the same behavior in this scenario.

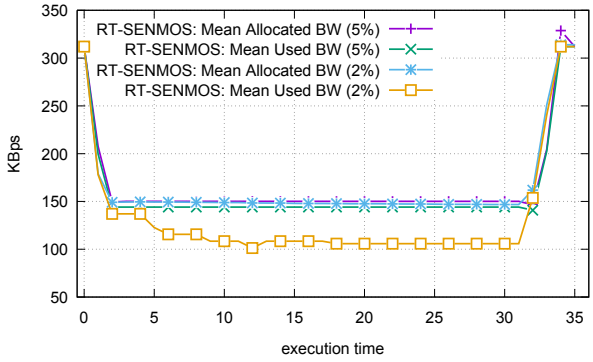
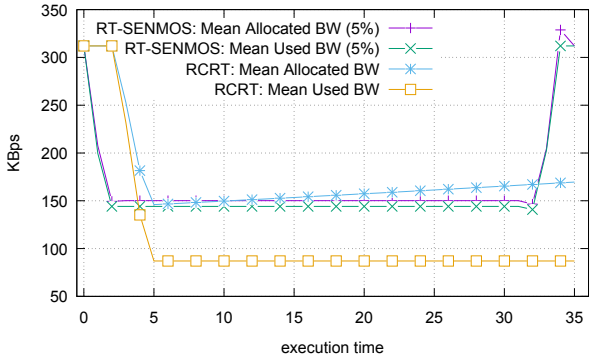
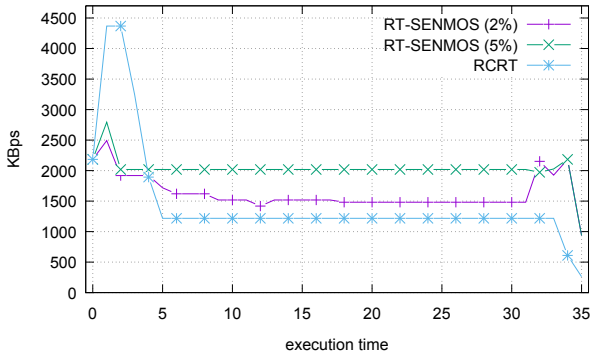


Fig. 8: Mean Allocated and Used Bandwidth (2% target loss).



(a) Mean allocated and used bandwidth.



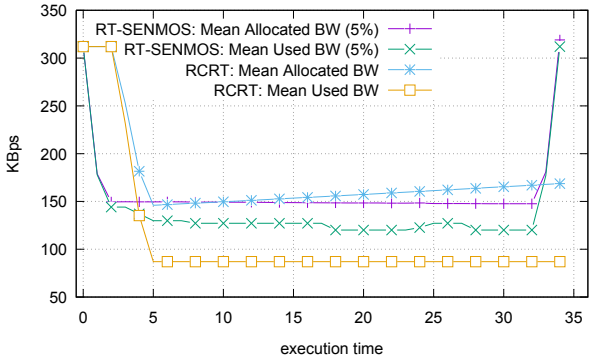
(b) Total used bandwidth.

Fig. 9: Bandwidth Allocation and Usage (2% target loss).

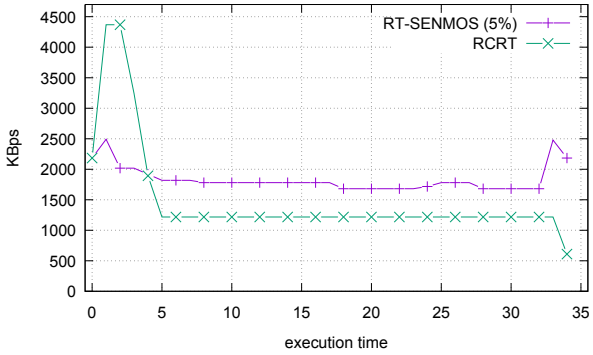
remains unused), without of course affecting the bandwidth actually used.

This is also reflected in Figure 9b, which shows the total bandwidth used by RCRT and RT-SENMOS with a 2% and a 5% loss rate threshold. RCRT is slow to detect the actual bandwidth available, as it relies on slow RTT measurements, causing an initial overallocation (it allocates 4.5 MBps, where only 2 MBps are available) which leads to congestion. Even in steady state, RCRT exploits less of the available bandwidth than RT-SENMOS, regardless of loss threshold.

Figure 10 shows the same metrics as Figure 9, with a target loss rate of 5% rather than 2%. The gap between allocated and used bandwidth per sensor is much higher for RCRT, as



(a) Mean allocated and used bandwidth.



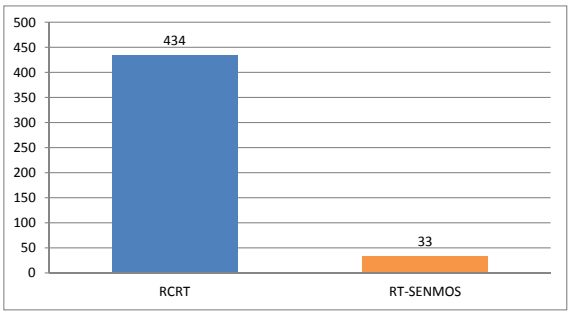
(b) Total used bandwidth.

Fig. 10: Bandwidth Allocation and Usage (5% target loss).

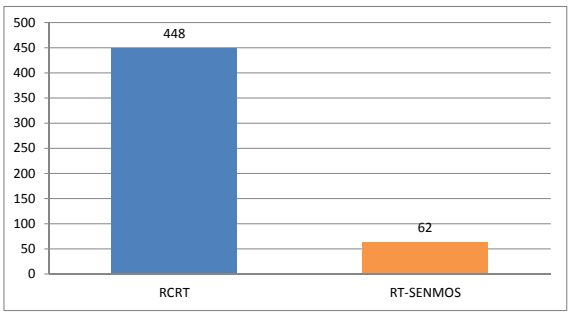
shown in Figure 10a, even though the used bandwidth for RT-SENMOS has dropped due to the higher loss rate used. Note again the gradual increase in the rates allocated by RCRT, which has no tangible benefit to actual bandwidth used. The total bandwidth used across all sensors, shown in Figure 10b, is much higher for RT-SENMOS, due to its quick adaptation and its awareness of the rate allocation requirements of the application; again, RCRT overallocates rates at the beginning, leading to congestion.

Finally, Figure 11 shows that RCRT generates far more control packets for rate allocation[4], as it sends rate allocation change messages to *all* nodes, in contrast to RT-SENMOS which controls each node separately. During the steady state period shown in Figures 9 and 10, RCRT detects that the network is under-utilized and tries to slowly increase the rates assigned to nodes. As it is not aware of the desired rates for the event sensors, these rate changes do not impact the real data transfers, but add a large traffic overhead. RCRT will only manage to upgrade the bit rate usable by those nodes after a long time and a very large number of rate control messages. With a 5% loss rate threshold and a 2% target loss rate, RCRT requires 13 times as many messages as RT-SENMOS, while with a 5% target loss rate the difference drops to 7 times as many messages. In the latter case, RT-SENMOS reaches the loss rate threshold and reconfigures the rates assigned to nodes, trying to strike a balance between high bandwidth utilization

[4]All other control messages are the same in both protocols.

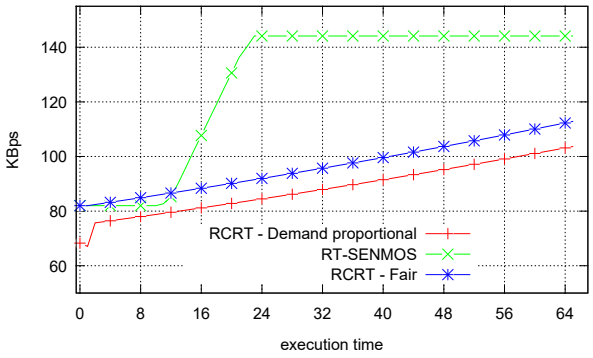


(a) 2% target loss.

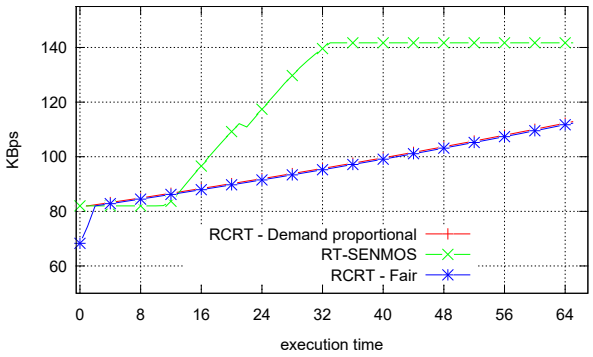


(b) 5% target loss.

Fig. 11: Rate Control Messages Sent (5% loss threshold).



(a) 2% target loss.



(b) 5% target loss.

Fig. 12: Mean Allocated Bandwidth (5% loss threshold).

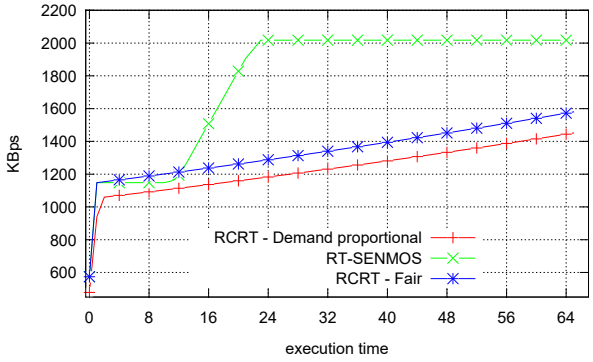
and limited loss rate. RCRT, on the other hand, always sends rate control messages to all sensors.

2) Periodic sensor scenario: In the *periodic sensor* scenario, sensors can use any bandwidth made available to them, starting at their indicated rate, unlike event sensors which operate only at specific rates. This allows us to compare the way each protocol exploits the available bandwidth without rate quantization effects. Note that with periodic sensors the allocated bandwidth is equal to the bandwidth used, therefore we do not separately show the bandwidth used. While this scenario is also supported by the first version of RT-SENMOS (see [4] for such results), the second version has been further optimized to make its behavior even more stable.

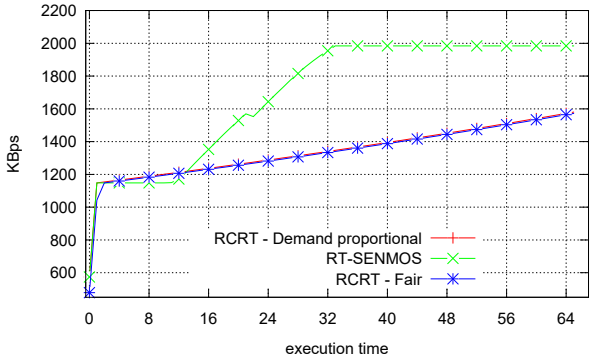
Figure 12 shows the rates allocated by RT-SENMOS with a loss rate threshold of 5% and two RCRT policies (fair and demand-proportional) at target loss rates of 2% and 5%. At the desired rate of 82 KBps, the system is uncongested, therefore all policies can increase their bandwidth allocations. While both RCRT policies operate similarly, since all sensors have asked for the same rate, making fair the same as demand-proportional, they only increase the rates slowly, as seen in the previous scenario; in contrast, RT-SENMOS quickly reaches the maximum available bandwidth and remains there for the rest of the experiment.

In Figure 13 we show the total bandwidth allocated to all sensors for the same experiment. It is clear that RT-SENMOS can quickly reach the maximum available bandwidth of 2 MBps, unlike the RCRT policies that do not manage to get even close to it until the end of the experiment. The change in the loss rate emulated does not make a big difference for the RCRT policies, since the system remains uncongested until the end of the run; as mentioned previously, the actual losses emulated are proportional to the congestion in the network. In contrast, RT-SENMOS quickly reaches the saturation point of the network, hence congestion does induce losses. The result is that in the 5% loss rate case RT-SENMOS takes slightly more time to reach its equilibrium state.

3) Mixed sensor scenario: We finally discuss the results from the *mixed* sensor scenario, where both sensor types co-exist. In this scenario, RT-SENMOS first assigns a fixed fraction of the available bandwidth to each sensor type, and then performs rate allocation separately for each sensor type. Figures 14a and 15a show the mean bandwidth allocated to periodic and event sensors, respectively, when we reserve 10%, 30% and 50% of the bandwidth for periodic sensors, with a loss rate threshold of 5% at a target loss rate of 2%. We can see that RT-SENMOS manages to balance the performance of the different sensor types depending on these allocations, thus allowing the application to implement priorities among sensor types. Each type of sensor quickly converges to a fair rate allocation: periodic sensors are all connected at the beginning, hence they quickly reach their fair shares, which increase as some of them complete their transmissions; event

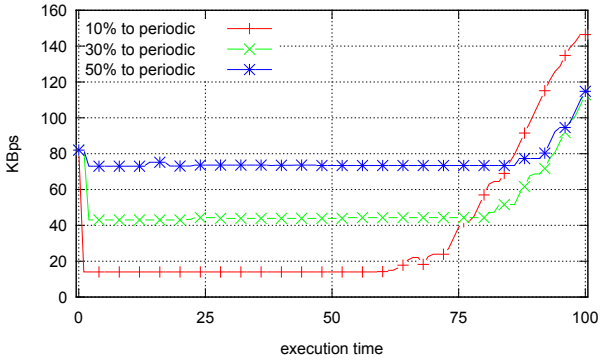


(a) 2% target loss.

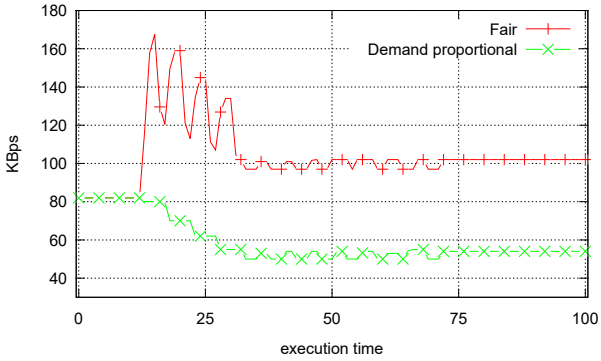


(b) 5% target loss.

Fig. 13: Total Bandwidth Used (5% loss threshold).



(a) RT-SENMOS.



(b) RCRT.

Fig. 14: Mean Bandwidth Allocated to periodic sensors (2% target loss).

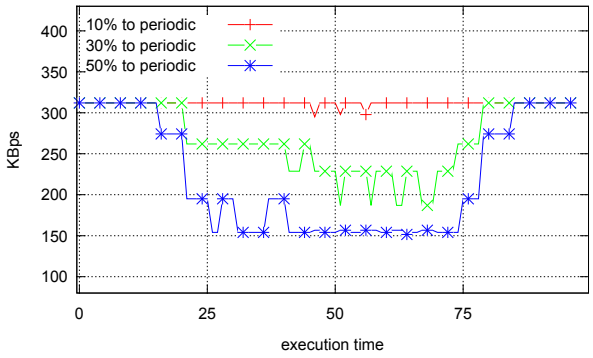
sensors are gradually connected, hence they start with larger allocations, which are reduced when a large number of them is connected, and are then increased as the first ones complete their transmissions. Note also the the continuous rate allocation for periodic sensors against the quantized rate allocation for event sensors.

Figures 14b and 15b show the same metrics for RCRT, using the fair and demand-proportional rate allocation policies. As event sensors join, the system quickly becomes congested and the two policies clearly show their differences. The fair policy of RCRT tries to fairly allocate the available bandwidth among all sensors, regardless of their desired rates, therefore all sensors eventually converge at roughly 100 KBps. This rate is too high for the periodic sensors and too low for the event sensors. The demand-proportional policy takes the desired rates into account, trying to equalize the fractions of the desired rates allocated to each sensor, therefore it allocates higher rates to the event sensors, but the resulting rates are again not ideal: periodic sensors get 50 KBps, which is less that the desired 82 KBps, and event sensors get 200 KBps, between the desired rates of 312 KBps and 187 Kbps. As a result, event sensors will leave some bandwidth unused, which cannot be used by the periodic sensors.

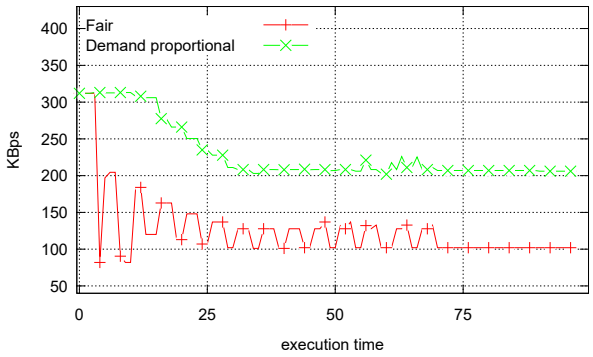
Unlike periodic sensors which can use any allocated rate, with event sensors only a specific set of rates is allowed, therefore the allocated bandwidths may exceed the used ones. As shown in Figure 16a, the mean bandwidth used by event sensors with RT-SENMOS is nearly exactly the same as the allocated bandwidth, shown in Figure 15a, while with RCRT the used bandwidth, shown in Figure 16b, is less than the allocated bandwidth, shown in Figure 15b. As in the event sensor scenario, with the fair policy the event sensors cab only use 87 KBps out of the 100 KBps allocated, while with the rate-proportional policy the event sensors can only use 187 KBps out of the 200 KBps allocated.

C. Discussion

While RT-SENMOS and RCRT are both sink-driven protocols, they have four critical differences. The first difference is that RCRT estimates congestion based on the time to recover a lost packet, a metric that takes time to converge. RT-SENMOS uses NAKs and timeouts to detect congestion early, measuring the RTT of the connection with control packet pairs. In scenarios with mobile rescuers, conditions around the sink change very fast, requiring the quick adaptation of RT-SENMOS. The second difference is that RCRT uses an AIMD scheme to allocate rates, which periodically leads to congestion. Congestion induces losses, which cause dramatic rate reductions. RT-SENMOS uses fixed increase and decrease steps for rate allocation. This leads to more stable behavior in scenarios with quick disconnections and congestion around the sink, as in emergency response applications. The third difference is that while RCRT uses different policies to allocate rates to sensors based on their bandwidth requirements, RT-SENMOS allows different types of sensors to separately

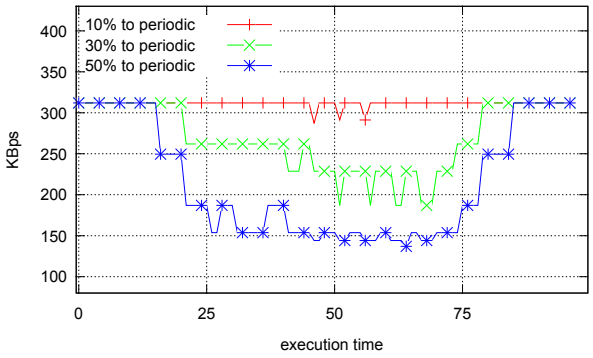


(a) RT-SENMOS.

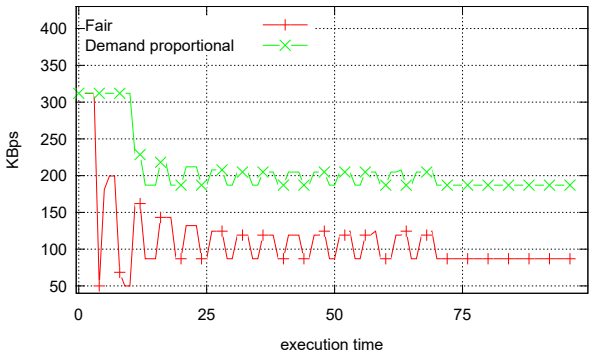


(b) RCRT.

Fig. 15: Mean Bandwidth Allocated to event sensors (2% target loss).



(a) RT-SENMOS.



(b) RCRT.

Fig. 16: Mean Bandwidth Used by event sensors (2% target loss).

manage their rate allocations, so as to guarantee that each category will receive a minimal amount of service. As the exact manner in which the available bandwidth should be split between sensors generally depends on the scenario, RT-SENMOS allows the application to fully tune each class to the underlying network, with no changes in the sensors, depending on the priorities of the emergency response scenario at hand. The fourth difference is that while RCRT sends original and retransmitted packets together, RT-SENMOS can either recover from errors with immediate retransmissions or in recovery rounds. The latter option allows the application to control the level of reliability required, by stopping the retransmissions whenever the received data are deemed sufficient; if, however, the application desires full reliability, it can continue the retransmission rounds as long as desired. This provides great flexibility in emergency response applications which can prioritize data depending on sensor type, again with no changes to the sensors.

VI. Conclusion

This paper presented and evaluated a reliable transport protocol for WSNs, RT-SENMOS, which was designed for emergency response applications where a mobile rescuer acts as the data sink. RT-SENMOS is implemented at the application layer and concentrates all policy decisions at the sink, thus allowing a set of generic sensors to be combined with a customized sink that can synthesize a specific protocol behavior by employing an existing set of error and congestion control tools. In our case study, this flexibility enabled a disaster recovery application to split the available bandwidth between different sensor types and between sensors of the same type depending on its preferences. RT-SENMOS controls each sensor individually, depending on the level of congestion, and allows different types of sensors to employ different loss recovery schemes. All these are possible by only modifying the sink, with no changes to the sensors.

We also provided a performance evaluation of RT-SENMOS against RCRT, using real implementations of both protocols in Java on a network with emulated losses and mobility. RT-SENMOS allocates all available bandwidth and tries to meet the actual requirements of each type, while RCRT utilizes 60% to 90% of the available bandwidth, depending on the rate allocation policy used. Our experiments show that the gains from RT-SENMOS are more pronounced with sensors that operate over a specific set of rates, such as video cameras. Furthermore, RT-SENMOS is quicker to adapt to network conditions than RCRT, reaching full bandwidth utilization faster than RCRT, which only gradually adapts to the available bandwidth. Finally, RT-SENMOS requires a much smaller number of control messages, as it sends rate control messages only to individual nodes and only when needed, unlike RCRT which constantly sends new rates to all nodes.

ACKNOWLEDGMENT

This research was co-financed by the European Union (European Social Fund) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework - Research Funding Program: THALIS - DISFER and by the RC-AUEB funded "Original Scientific Publications" project under contract ER-2766-01.

REFERENCES

[1] C. Stais, G. Xylomenos, and E. Zafeiratos, "RT-SENMOS: Sink-driven congestion and error control for sensor networks," in *Proc. of the IFIP Conference on New Technologies, Mobility and Security (NTMS)*, 2016.

[2] J. Paek and R. Govindan, "RCRT: Rate-controlled reliable transport for wireless sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2007, pp. 305–319.

[3] C. Stais, G. Xylomenos, and G. F. Marias, "Sink controlled reliable transport for disaster recovery," in *Proc. of the ACM Conference on Pervasive Technologies Related to Assistive Environments (PETRAE)*, 2014, pp. 1–4.

[4] C. Stais and G. Xylomenos, "RT-SENMOS: Reliable transport for sensor networks with mobile sinks," in *Proc. of the IEEE Symposium on Computers and Communication (ISCC)*, 2015, pp. 105–110.

[5] F. Stann and J. Heidemann, "RMST: reliable data transport in sensor networks," in *Proc. of the IEEE Workshop on Sensor Network Protocols and Applications (SNPA)*, 2003, pp. 102–112.

[6] Y. G. Iyer, S. Gandham, and S. Venkatesan, "STCP: a generic transport layer protocol for wireless sensor networks," in *Proc. of the IEEE Conference on Computer Communications and Networks (ICCCN)*, 2005, pp. 449–454.

[7] B. Deb, S. Bhatnagar, and B. Nath, "ReInForM: reliable information forwarding using multiple paths in sensor networks," in *Proc. of the IEEE Conference on Local Computer Networks (LCN)*, 2003, pp. 406–415.

[8] O. B. Akan and I. F. Akyildiz, "Event-to-sink reliable transport in wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, pp. 1003–1016, Oct. 2005.

[9] C. Wang, K. Sohraby, V. Lawrence, B. Li, and Y. Hu, "Priority-based congestion control in wireless sensor networks," in *Proc. of the IEEE Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, 2006.

[10] A. Woo and D. E. Culler, "A transmission control scheme for media access in sensor networks," in *Proc. of the ACM Conference on Mobile Computing and Networking (MobiCom)*, 2001, pp. 221–235.

[11] C.-Y. Wan, S. B. Eisenman, A. T. Campbell, and J. Crowcroft, "Siphon: Overload traffic management using multi-radio virtual sinks in sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005, pp. 116–129.

[12] J.-Y. Teo, Y. Ha, and C.-K. Tham, "Interference-minimized multipath routing with congestion control in wireless sensor network for high-rate streaming," *IEEE Transactions on Mobile Computing*, vol. 7, no. 9, pp. 1124–1137, 2008.

[13] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, "CODA: Congestion detection and avoidance in sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003, pp. 266–279.

[14] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 134–147.

[15] C. T. Ee and R. Bajcsy, "Congestion control and fairness for many-to-one routing in sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 148–161.

[16] K. Karenos and V. Kalogeraki, "Traffic management in sensor networks with a mobile sink," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 10, pp. 1515–1530, 2010.

[17] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *Proc. of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM)*, 1990, pp. 200–208.

[18] C. Stais, G. Xylomenos, and A. Voulimeneas, "A reliable multicast transport protocol for information-centric networks," *Journal of Network and Computer Applications*, vol. 50, pp. 92–100, 2015.

[19] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance," *Computer Networks and ISDN systems*, vol. 1, pp. 1–14, 1993.

[20] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, 1993.