RT-SENMOS: Reliable Transport for Sensor Networks with Mobile Sinks

Charilaos Stais and George Xylomenos
Mobile Multimedia Laboratory
Athens University of Economics and Business
Athens, Greece +30 210 8203693
Email: stais@aueb.gr, xgeorge@aueb.gr

***Abstract*—Gathering information efficiently from a, possibly fragmented, sensor network presents a serious problem in disaster recovery applications. Unless a transmission control mechanism exists, a sink can be flooded with information sent by sensors, or sensor transmissions may be lost on their way to the sink. Consequently, there is a necessity for a reliable protocol that automatically and speedily adapts to losses, congestion and network changes due to sink mobility. This paper describes RT-SENMOS, a reliable transport protocol for controlling sensor transmissions based on sink-assigned rates. The sink decides how to share the available bandwidth among the sensors and also determines the reliability to be achieved in each case. Our protocol operates on top of UDP/IP, therefore it can be directly integrated into a disaster recovery application that will set its parameters depending on the situation. Moreover, as it is fully sink-controlled, it enables the use of simple and inexpensive fixed sensors, which offload all protocol intelligence to a more expensive but reusable mobile sink. We present the design of the protocol, comparing it with similar approaches, and evaluate its performance using a real implementation.**

I. Introduction

In-building sensor networks can provide great assistance to human or robotic rescuers in disaster recovery situations. In such scenarios, a rescuer roams a disaster area, for example, a building hit by fire or an earthquake, gathering information from any surviving sensors: temperature sensors can indicate whether nearby areas are on fire, chemical sensors can detect the presence of people breathing, and audio and video sensors can reveal what is behind a blocked passage. As part of the *DIstributed Sensor systems For Emergency Response* (DISFER) project,[1] we aim to advance the state of the art in disaster recovery using sensor network technologies to improve the capabilities of the rescuers.

In such disaster recovery situations, the sensor network may be partially connected, as the disaster may have wiped out parts of the fixed infrastructure. However, the mobile rescuer can come into contact with most of the sensors, either directly, or via other sensors in a multi-hop configuration, while roaming. This means that the transport protocol used to transfer data from the sources (sensors) to the sink (rescuer) must establish connections and exchange information quickly and reconfigure itself efficiently. Moreover, it must be (mostly) reliable, since the rescuer cannot count on sensor redundancy for gathering critical information. Lastly, the transport protocol must avoid losing packets due to congestion around the sink, where data from many sources naturally converges.

A crucial observation is that all the aforementioned requirements revolve around the sink: the sink must reliably receive data, sink mobility makes the network view change, and it is the area near the sink that is most likely congested. We have therefore designed the *Reliable Transport protocol for SEnsor Networks with MObile Sinks* (RT-SENMOS), a novel, purely sink-controlled protocol, in the sense that the sink allocates transmission rates to all reachable sources and manages the error recovery process depending on application objectives. In previous work we presented the design of RT-SENSMOS [1]; in this paper, we review its design and its implementation details, and present a preliminary performance evaluation of a real RT-SENMOS implementation over an actual network.

The remainder of this paper is organized as follows. In Section II we present our assumptions and motivate our design choices. In Section III we review protocol operation and in Section IV we explain the rate control scheme used to handle congestion. Section V describes our implementation, while its performance is evaluated in Section VI. We discuss related work and contrast it with our protocol in Section VII. We conclude and discuss our plans for future work in Section VIII.

II. Assumptions and Rationale

RT-SENMOS assumes a set of fixed sensors forming a multi-hop network using a shared communication channel, such as WiFi, Bluetooth or ZigBee. Each sensor acts as a source attempting to transmit data to a mobile sink, i.e. a human or robotic rescuer equipped with a mobile computer. The sensors have been pre-programmed to send the data to the sink, that is, they know the network address of the sink. At any given time, the sink may be able to communicate with only a subset of the sensors, since some may not be reachable at that time. Additionally, it is assumed that an underlying protocol routes data between all nodes in the current network. A routing protocol based on the *received signal strength* (RSSI) metric, available in most wireless interfaces, which triggers route recalculations whenever the sink detects that the node with the highest RSSI has changed [2], is suitable.

We also assume that there are multiple types of sensors, which may need to be treated separately. The sink dedicates

[1] http://www.aueb.gr/disfer

a portion of its available bandwidth to each sensor type and then assigns a part of that bandwidth to each individual sensor of that type. Dividing the sensors into type groups allows disaster recovery applications to decide how to share the available bandwidth between groups, as well as to operate different bandwidth allocation schemes for each group. In RT-SENMOS, all sensors are treated as data sources that may alternate between active and idle states; when active, they require a specific transfer rate. In our test scenario, sensors are divided into two groups: event and continuous sensors. An *event sensor* collects point data, e.g. a temperature value or a camera snapshot, while a *continuous sensor* sends a continuous data stream, e.g. live video or audio. Event sensors send data periodically and not only when, for example, the temperature changes or a movement is detected, since the sink needs to gather as much data as possible over the, possibly limited, period during which each sensor is reachable.

For congestion control, the sink *explicitly* controls the transmission rate of all sources. As all source transmissions converge at the sink, the latter is in a position to detect the onset of congestion and take measures to restrain it. The sink is also aware of application requirements, that is, the sink knows which of the sensors are of greater importance, and therefore appropriately regulates their transmission rates depending on the situation. This centralized approach makes sensors simpler, cheaper, and with longer battery life. Distributed congestion control would be very complex in our setting, since reachable sensors change all the time. In contrast, the sink is always in the center of the reachable network, while sources seem to join and leave the network.

In terms of reliability, our protocol uses *negative acknowledgments* (NACKs) to trigger the retransmission of lost data. Most protocols retransmit lost data immediately, but RT-SENMOS retransmits lost data in so-called recovery rounds. First, all the data packets are transmitted. Next, the data packets for which NACKs have been received are retransmitted, then the ones for which NACKs have been received again are retransmitted, and so on. This allows the sink to stop the recovery process whenever it deems appropriate, for instance, when enough packets have been received to reconstruct the content. Furthermore, if a source loses connectivity with the sink before the transmission of the data packets completes, the sink can approximately reconstruct the content, as it will have received incomplete data from beginning to end, rather than complete data only from beginning.

For rate allocation, the sink can implement any desired scheme, independently from the sensors. In our implementation, the sensors are divided into classes according to their type, and separate rate allocation is performed for each class, prioritizing the event sensors. The rationale behind our rate allocation policy is that although higher transmission rates make continuous sensors more useful by, for example, providing more informative videos, continuous sensors should be restrained to allow the low-bandwidth event sensors to transmit data all the time. However, the way the available bandwidth is shared depends on the application and can be adjusted to suit the needs of a specific rescue mission.

Our protocol executes at the application-layer over UDP/IP and thus can be implemented for any device with IP connectivity and a UDP socket interface. It is also written in Java, allowing it to use multiple devices as mobile sinks, e.g. Android smartphones and tablets, without requiring kernel modifications or root privileges. In addition to portability and ease of debugging, the protocol can be integrated into the application in order to directly control protocol parameters, such as the rate allocation to sensor types and the level of reliability required. For instance, in a building with a few sensors and many cameras, more bandwidth could be allocated to continuous sensors before the rescuer enters the building.

III. Protocol Description

In this section, we describe the five stages of communication between the sink and the sensors in RT-SENMOS: connection establishment, sensor information exchange, data exchange, idle and connection release. For devices alternating between active and idle periods, such as event sensors that periodically send data, the data exchange and idle periods are repeated.

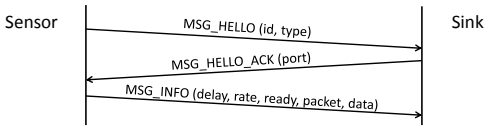

Fig. 1. Connection establishment and sensor information exchange.

A. Connection establishment

The RT-SENMOS model defines two separate communication channels per sensor, one for control data and one for user data. An additional common control channel attached to a well-known UDP port is used for connection requests. This approach allows the sink to dedicate one thread to listening for connection requests and a set of threads for exchanging control plane messages with sensors, adding data channel threads on demand. As a result, there is no need for multiplexing control messages from multiple sensors over a single channel.

When the sink begins to operate, it listens to a well-known UDP port for connection requests from the sensors. The sensors wait for the sink to become reachable before trying to connect to it. As shown in Figure 1, each sensor may periodically send a probe message, `MSG_HELLO`, to the well-known IP address and UDP port of the sink until it receives a response. Alternatively, the routing protocol used may notify the sensor when the sink becomes reachable. The `MSG_HELLO` message includes the sensor identifier and its type, i.e. event or continuous sensor. The sink responds to the `MSG_HELLO` message with a `MSG_HELLO_ACK` message. The `MSG_HELLO_ACK` message indicates a UDP port dedicated to that sensor for its control messages. If the sink does not have a pre-configured sensor identifier, then it leaves the corresponding field empty in the `MSG_HELLO` message and the sink assigns it one in the `MSG_HELLO_ACK` message.

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 to be used 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 MSG_HELLO_ACK, subtracting the delay in the message to get the *round-trip time* (RTT) to that sensor. At this point, the connection has been established and the sink is aware of the sensor's bandwidth requirements.

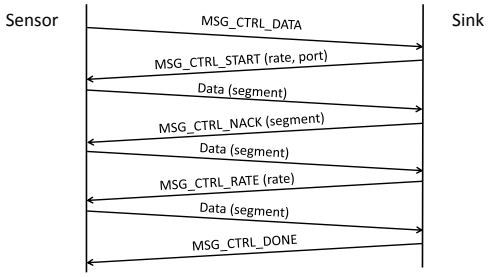

Fig. 2. Data exchange stage.

B. Data exchange and idle

If the sensor has indicated that it is not ready to send data in its MSG_INFO message, the sink moves to an idle state, waiting until the sensor sends a MSG_CTRL_DATA with no parameters through the control channel for that sensor, as shown in Figure 2. 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 data 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 initial rate allocated to the sink is set as explained in Section IV.

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, possibly long, queue of data messages. The sensor breaks down its transmission into packets of the size indicated in the MSG_INFO message, until the data indicated in the MSG_INFO message are 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. However, the sensor does not immediately retransmit lost messages. After the transmission is complete, all missing messages are retransmitted, generating further NACKs from the sink, if needed. This procedure is repeated in rounds, until all messages are received [3]. Once recovery is complete, the sink sends a MSG_CTRL_DONE message over the control channel to indicate a successfully completed transfer. Both endpoints then move to an idle state, until the sensor generates a new MSG_CTRL_DATA message. If needed, the sink may send a MSG_CTRL_RATE message to the sensor indicating its new rate allocation, as explained in Section IV.

Note that round-based recovery allows the sink to use any packets received without waiting for retransmissions. The sink may even prematurely stop the recovery process by sending the MSG_CTRL_DONE message. For example, when an image is transmitted using redundancy coding, the sink may stop the recovery process when enough packets have been received to adequately reconstruct the image. This allows the application to fine tune the reliability of the protocol.

Fig. 3. Connection control and release stage.

C. Connection control and release

Since the path between the sink and the sensor may become disconnected due to sink 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, as shown in Figure 3, 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 any side wishes to complete 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.

IV. Congestion management

The congestion management mechanism of RT-SENMOS is agile and purely sink-driven. Since we focus on congestion around the sink, we know the total available bandwidth, as it depends on the technology used by the sink and sensors for data exchange. We first reserve a fixed part of this bandwidth, e.g. 10%-30%, for the control message exchanges which are *not* rate-controlled. Then, the sink splits the remainder between event and continuous sensors using a ratio determined by the application, e.g. 10%-90%, depending on the number and type of sensors present at the disaster site.

The congestion management algorithm periodically evaluates the state of individual connections and the system as a whole. The sink monitors the RTT of each connection using the MSG_HELLO_ACK, MSG_INFO, MSG_CTRL_ALIVE and MSG_CTRL_ACK messages; the processing delay at the sensor is always subtracted to get an accurate RTT estimate. The congestion management algorithm maintains the last few RTT samples and their moving average.

Whenever the algorithm runs, it first checks whether the average for each sensor has increased compared to the previous

value by more than a configurable threshold. If this occurs four times in a row, then the corresponding connection is congested, otherwise it is not. If the connection is congested, the sink instructs the sensor to reduce its rate by 20%, via a `MSG_CTRL_RATE` message.

After each individual sensor is checked, if a new sensor has been connected or an existing one has been disconnected, the entire system is checked to see whether global adjustments need to be made. This takes places separately for each sensor class. First, we calculate the total rates requested (not assigned) by the sensors of the class. If these are below the available bandwidth, they will all get what they asked for. New sensors will get their requested rate in the `MSG_CTRL_START` message which directs them to start sending data. Sensors that were previously rate limited, will increase their rate by 20%, while other sensors will get their requested rate; in both cases, the change is announced via a `MSG_CTRL_RATE`. If, however, the requested bandwidth is higher than the available one, the available rate is shared *equally* among all sensors of that class. The sensors are notified as above, i.e. either via a `MSG_CTRL_START` or via a `MSG_CTRL_RATE` message.

The congestion management algorithm is very simple, as we expect congestion to be concentrated around the sink. Since the sink is constantly on the move, it is very unlikely that a distributed congestion control management will have time to converge. While TCP and many other transport protocols use an *Additive Increase - Multiplicative Decrease* (AIMD) algorithm, our scheme uses fixed and symmetric steps. This is because TCP sources constantly probe the network for capacity, hence entering deep into the congested region before having to abruptly backoff. In our scheme sensors are conservatively rate controlled, hence congestion is expected to appear slowly, thus avoiding the need for dramatic rate reductions.

V. Implementation

Our RT-SENMOS prototype was implemented in Java, requiring 21 KB of bytecode for the sensor and 37 KB of bytecode for the sink. The implementation runs entirely at the user level, allowing it to be compiled jointly with the application that uses it, taking advantage of the API provided by RT-SENMOS. In addition, the sensor side has been ported to the Android OS, as most Android smart phones have specifications that make them suitable for acting as sensors. The prototype uses configuration files to set the behavior of the protocol, for example, the shares of bandwidth between the sensor categories; these can instead be set directly by the application. Similarly, the prototype uses files stored on disk in lieu of actual sensor data; these can instead be objects generated by the sensor. A well-known UDP port and IP address needs to be agreed between the sink and sensors to allow them to rendezvous, but the additional control and data ports for each sensor are assigned automatically by the protocol. The protocol does not require any changes to the kernel or the libraries of the operating system, or even superuser access, since it operates over simple UDP/IP sockets.

TABLE I
Experimentation parameters

Parameter	Value
Content size (MB)	8
Requested bit rates (KBps)	82 (event) or 200 (continuous)
Chunk size (bytes)	512
RTT samples stored per sensor	10
Bandwidth available at sink (Mbps)	16, 40 or 54
User bandwidth share	70% or 90%
Reserved sensor share	10%
Loss probability	2% or 5%

VI. Performance evaluation

A. Experimental setup

To evaluate the performance of our RT-SENMOS, we relied on a deployment of sensors on 28 nodes, 14 event and 14 continuous, with one node acting as a sink, all connected over a single hop. As our main target was to preliminary evaluate the performance of RT-SENMOS, the adverse radio environment was emulated by artificially injecting packet losses of either 2% of 5% and the sink was located at a fixed spot. Each sensor node, regardless of its class, had 8 MB of data to send to the sink in total, in 512 byte chunks. We assumed that event sensors sent screenshots at a rate of 82 KBps, while continuous sensors streamed video at 200 KBps. Experiments were executed with the bandwidth available at the sink being 16, 40 and 54 Mbps. Sensors started transmitting one after the other, with a 1 sec delay between them, and we gathered data until all sensors had completed their transmissions. We reserved either 10% or 30% of the available bandwidth for control data, and split 10% of the remainder to event sensors and the rest to continuous sensors.

B. Experimental results

We first recorded the actual resource needs for both the sensor and sink packages. These values include the resource consumption of a standard Java Virtual Machine and all code and data memory consumed. Concerning memory usage, the sink needs 283 MB on average (min: 147 MB, max: 377 MB), while the sensors only need 10 MB. The sink required a mean processing power of 427 MHz on a standard Intel architecture CPU, while the sensors were separated based on type – event sensors needed 102 MHz while continuous ones needed 306 MHz. We observe that the sink needs are much greater than those of the sensors, due to the fact that the sink has to control multiple sensors simultaneously, receive data, check for congestion, send control messages and handle sensor connections and disconnections. On the other hand, a typical sensor can support the operation of both types of sensor package. For example, a Raspberry Pi with a quad core 900 MHz CPU and 1 GB of RAM can serve even as a sink.

We then evaluated the effectiveness of the RT-SENMOS sink in rate controlling the sensors. Figures 4 and 5 show the average allocated rate for event and continuous sensors, respectively, against time, for an experiment with an available bandwidth of 16 Mbps, a 5% loss and 70% of the bandwidth available to sensor data. Note that the timescales are not linear, since we only take rate samples whenever the sink changes

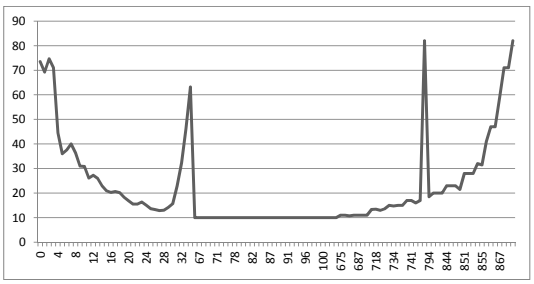

Fig. 4. Average event sensor bandwidth over time (KBytes) at 16 Mbps, 5% loss and 70% user bandwidth share.

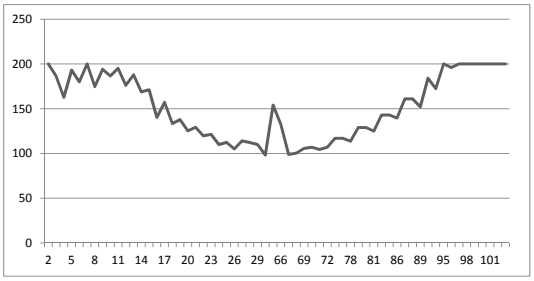

Fig. 5. Average continuous sensor bandwidth over time (KBytes) at 16 Mbps, 5% loss and 70% user bandwidth share.

some rates, and that the two figures have different timescales, since continuous sensors finish earlier due to their higher rates. Based on the above experimental parameters, the bandwidth available for user data is 11.2 Mbps, split to 1.1 Mbps to event sensors and 10.1 Mbps to continuous sensors. Since there are 14 sensors of each type, the available rates per sensor are 10 KBps for event and 90 KBps for continuous sensors.

We observe that in both event and continuous sensors, the average allocated bandwidth starts from the highest possible value (82 KBps and 200 KBps respectively) and decreases as sensors connect to the sink and start data transmission. In Figure 4, this average rate stabilizes for a long period at 10 KBps, the guaranteed event sensor rate calculated above, as the pool of sensors remains unchanged and rates are allocated equally to all sensors. As Figure 5 shows, continuous sensors also converged to their expected rates. As the continuous sensors start disconnecting due to data exchange completion, the average continuous sensor bandwidth increases. When they all complete, the event sensor bandwidth also starts increasing. The peaks we observe at a few points represent attempts of the mechanism to slowly increase rates by getting advantage of RTT stability. Unfortunately, this leads to congestion and triggers the rate re-allocation algorithm, which leads all rates back to their predicted points. Results from experiments with different parameters are substantially the same as these.

We then examined the efficiency of the RT-SENMOS transport proper. Figure 6 shows the average completion time with 2% loss rate with different bandwidth and user bandwidth shares, broken down into initial distribution time, i.e. before losses are repaired, and recovery time, i.e. when data packets are retransmitted in rounds triggered by NACKs, while Figure 7 shows the same metric with a 5% loss rate. As expected, completion time is reduced with higher overall bandwidth and higher user bandwidth shares. We can also see that the recovery time is a small fraction of the initial distribution time, even though it grows with a higher loss rate, indicating that recovery in RT-SENMOS is very quick.

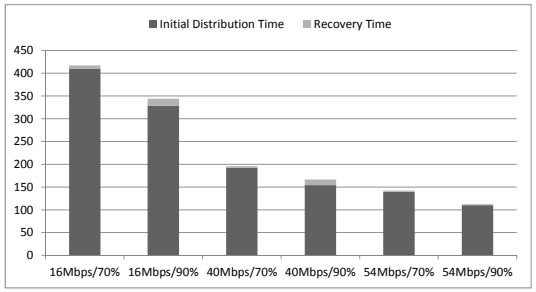

Fig. 6. Completion time (2% loss, various bandwidths and user shares).

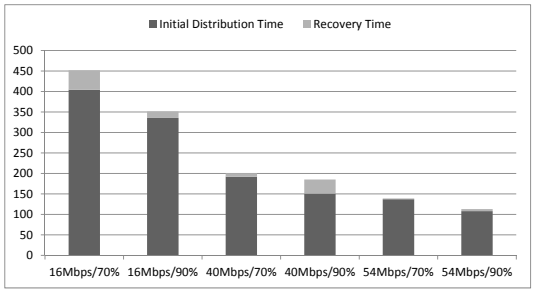

Fig. 7. Completion time (5% loss, various bandwidths and user shares).

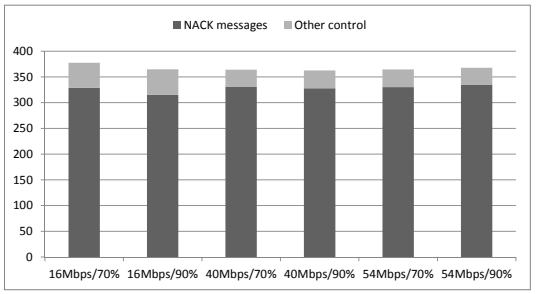

Fig. 8. Control overhead (2% loss, various bandwidths and user shares).

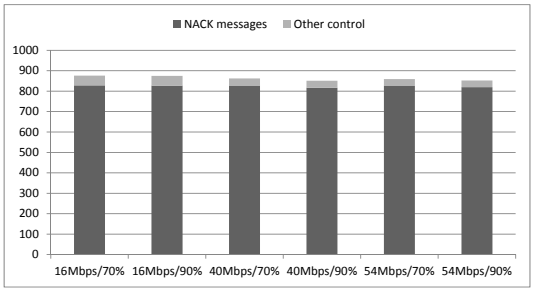

Fig. 9. Control overhead (5% loss, various bandwidths and user shares).

Figure 8 shows the control overhead for the same set of experiments with a 2% loss rate, broken down into NACKs and all other control messages. It is clear from the figure, that the non recovery related control overhead of RT-SENMOS is minimal, as most of the control messages are NACKs triggered by losses. At a 5% loss rate, the control overhead, as shown in Figure 9, roughly doubles due to the higher loss rates, but the non recovery related control overhead remains fixed, thus representing a smaller fraction of the overall control overhead, indicating the control efficiency of RT-SENMOS.

VII. Related Work

According to the taxonomy in [4], transport protocols can be classified based on their approach to reliability and congestion control: a protocol may offer unreliable or reliable service and a protocol may offer no congestion control, distributed congestion control or centralized congestion control. Reliability can be further subdivided to hop-by-hop reliability via retransmissions, as in RMST [5], end-to-end reliability via retransmissions, as in RCRT [4] and STCP [6], and forward reliability without retransmissions, as in ReInForM [7]. RT-SENMOS implements reliable service, achieved via end-to-end retransmissions, and centralized congestion control.

While the hop-by-hop reliability of RMST is useful for a wireless environment, most wireless networks can retransmit lost packets at the link layer [5]. RT-SENMOS therefore concentrates on congestion induced losses, as in RCRT and STCP, which we handle end-to-end. The use of multiple transmissions without NACKs, as in ReInForM, requires a well-connected sensor network, which is unlikely in disaster recovery applications. For congestion control, we chose a centralized approach as in RCRT for many reasons. First, as data converge at the sink, packet drops will occur if the sensors are not regulated. Second, in our target application the sink is mobile, therefore distributed congestion control would probably not converge. Third, by concentrating congestion control decisions at the sink as in [4], we can modify its behavior depending on the environment and application. Fourth, this approach does not require sensors to implement any congestion control measures as in [6].

RT-SENMOS is most similar to RCRT [4], as it is based on the sink explicitly controlling the transmission rates of the sensors and sending NACKs to the sensors. The differences between RCRT and RT-SENMOS are due to the fact that we explicitly address mobile sinks, as in COSMOS [2], therefore we have implemented simpler and faster control loops than RCRT, while avoiding the distributed congestion control of COSMOS which we expect to be slow to converge. On the other hand, while both STCP and ReInForM provide limited reliability, RT-SENMOS allows the sink to dynamically define and control the reliability level depending on the application, unlike ReInForM where the reliability goal is fixed when a packet is generated [7] and STCP where the sink controls reliability but the sensors set the reliability goals [6].

VIII. Conclusion and Related work

We have presented a reliable transport protocol for sensor networks, RT-SENMOS, especially suitable for disaster recovery applications. RT-SENMOS assumes that the sink is mobile, thus requiring a fast and agile method for congestion control. RT-SENMOS is purely sink driven and implemented at the application layer, thus allowing application policies to be set at the sink without previously configuring the sources. Furthermore, it allows bandwidth to be split between different classes of sensors and within each class depending on application preferences. Finally, it adjusts the allocated bandwidth to the current state of the network by taking into account RTT measurements to detect the onset of congestion and applying rate changes to individual sensors. Our measurements from a real RT-SENMOS implementation indicate that the protocol is effective in enforcing the desired rate allocations, while its control overheads are low and its completion times reasonable.

Future work includes a performance comparison of RT-SENMOS against our own implementation of RCRT, using a multi-hop network and a mobile sink, so as to more closely emulate an actual disaster recovery scenario. Another direction is modifying the rate control algorithm to allocate unused bandwidth from one sensor type to other types.

Acknowledgment

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS - Athens University of Economics and Business - DISFER.

References

[1] C. Stais, G. Xylomenos, and G. F. Marias, "Sink controlled reliable transport for disaster recovery," in *Proc. of the ACM PETRA*, 2014.

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

[3] C. Stais, G. Xylomenos, and A. Voulimeneas, "A reliable multicast transport protocol for information-centric networks," *Journal of Network and Computer Applications*, 2014.

[4] J. Paek and R. Govindan, "RCRT: Rate-controlled reliable transport for wireless sensor networks," in *Proc. of SenSys '07*, 2007, pp. 305–319.

[5] F. Stann and J. Heidemann, "RMST: reliable data transport in sensor networks," in *Proc. of the IEEE SNPA Workshop*, 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 ICCCN*, 2005, pp. 449–454.

[7] B. Deb, S. Bhatnagar, and B. Nat, "ReInForM: reliable information forwarding using multiple paths in sensor networks," in *Proc. of the IEEE LCN*, 2003, pp. 406–415.