Performance Evaluation of DTSN in Wireless Sensor Networks

Francisco Rocha[1], António Grilo[1,2], Paulo Rogério Pereira[1,2], Mário Serafim Nunes[1,2], and Augusto Casaca[1,2]

[1] INESC, Rua Alves Redol, nº 9, Lisboa, Portugal
[2] IST, Av. Rovisco Pais, Lisboa, Portugal
rocha.francisco@gmail.com, antonio.grilo@inov.pt, prbp@inesc.pt, mario.nunes@inov.pt, augusto.casaca@inesc.pt

Abstract. The guaranteed delivery of critical data is an essential requirement in most Wireless Sensor Network (WSN) applications. The paucity of energy, communication, processing and storage resources in each WSN node causes the TCP transport model (widely used in broadband networks) to be inefficient in WSNs, a reason why new WSN-specific reliable transport protocols have been proposed in the past few years. This paper presents one of these protocols, the Distributed Transport for Sensor Networks (DTSN). DTSN is able to efficiently support unicast communications in WSNs due to its capabilities to tightly control the amount of signaling and retransmission overhead. The basic loss recovery algorithm is based on Selective Repeat ARQ, employing both positive and negative acknowledgements. Caching at intermediate nodes is used to avoid the inefficiency typical of the strictly end-to-end transport reliability commonly assumed in broadband networks. DTSN is currently implemented in TinyOS. Preliminary simulation results using this code show that DTSN is quite efficient providing block oriented reliability, while the caching mechanism employed in DTSN decreases packet delay for more than one hop.

Keywords: Energy-efficiency, Reliable Data Transport, Quality of Service, Wireless Sensor Networks.

1 Introduction

The guaranteed delivery of critical data is an essential requirement in most Wireless Sensor Network (WSN) applications. Illustrative examples are: battlefield surveillance, intrusion detection and E-health monitoring applications, where critical alerts must be timely and reliably delivered to the monitoring stations that act on those data; and industrial control applications, where commands must be timely and reliably delivered to the actuators (e.g., robotic arm). Moreover, while in the beginning WSNs were thought to convey very simple data such as alerts, actuator commands and physical measurements, the recent availability of low-cost miniaturized hardware such as CMOS cameras and microphones have led to the possibility of transmission of larger data blocks filled with sound samples and/or images. Given the paucity of resources and high bit error rate (BER) values typically

L. Cerdà-Alabern (Ed.): Wireless and Mobility 2008, LNCS 5122, pp. 1–9, 2008.

featured by these networks, bulky data delivery can only be effected if the bandwidth utilization is maximized. This entails minimizing the end-to-end packet loss ratio (PLR), which is a traditional responsibility of the transport function.

Proven transport protocols like TCP [1], designed to support user applications in infrastructure networks, usually present significant inefficiencies when employed without considerable modification in WSN systems. Several proposals have been made for alternative reliable transport protocols, but only a few of them are suited to support block-oriented data delivery. The shortcomings found in these existent proposals led to the development of the Distributed Transport for Sensor Networks (DTSN) protocol [2], which is based on the following principles:

- Reliable transmission of block-oriented data;
- Energy-efficiency: to avoid useless wasting of energy resources through minimization of the control and retransmission overhead;
- Distributed Functionality: to exploit the WSN storage resources in a cooperative way, so that the scalability of WSN operation in a multihop environment is increased.

This paper presents a performance evaluation of DTSN based on its TinyOS [3] implementation.

The rest of the paper is structured as follows. Section 2 presents the most relevant related work; section 3 describes the DTSN protocol; section 4 presents the simulation results; section 5 concludes the paper and lists some topics of ongoing and future work.

2 Related Work

There are several proposals of reliable transport protocols for WSNs. This section only covers the proposals for block-oriented reliable transport that were considered more relevant for the development of DTSN.

Reliable Multi-Segment Transport (RMST) [4] was designed to work on top of the Directed Diffusion [5]. It offers two simple services: data segmentation/reassembly and guaranteed delivery. RMST can operate end-to-end or in a store-and-forward mode where intermediate nodes care to receive all the fragments of a block before forwarding it. RMST is affected by various weaknesses that make it somewhat unreliable and inefficient. First of all, there is no full guarantee of delivery. Since it only uses negative acknowledgements (NACKs) for loss recovery, when none of the fragments of a data block are received by an intermediate or destination node, the loss can not be detected because the sink is not aware of the transmission and there is no end-to-end positive ACK to finalize the transaction.

Distributed TCP Caching (DTC) [6] is a TCP enhancement to make it more efficient in WSNs. It improves the transmission efficiency by compressing the headers and the use of caching at selected nodes in the path from source to destination, thus minimizing end-to-end retransmissions. DTC is fully compatible with TCP, leaving the endpoints of communication unchanged – it only requires changes in the logic of intermediate nodes. The use of caching significantly improves the efficiency of packet loss recovery, minimizing the energy spent with

retransmissions. However, like in TCP, DTC features an inefficiency associated with the transmission of unnecessary positive ACKs, which is totally controlled by the receiver.

Pump Slow Fetch Quickly (PSFQ) [7] is a protocol primarily designed to offer downstream multicast guaranteed delivery for dynamic code update, though it can also be used for upstream unicast communication. It supports the reliable dissemination of consistent data blocks along the network within a user defined scope. Its behavior fluctuates between multihop forwarding during the data dissemination phase (Pump phase), and store and forward when losses are detected. Upon loss detection, forwarding is stopped and recovery phase (Fetch phase) is started. Data is disseminated through broadcasting during a pre-defined time interval at the end of which flow control is applied. That approach allows to stop the propagation of gaps in the fragment sequence as soon as possible and to quickly recover missing fragments from neighbors. Every node, in fact, stores the received data to be able to provide missing fragments to requesting neighbors. The use of flow control at every hop introduces a high delay in the communication. Besides, the fact that PSFQ only employs NACK feedback (like RMST) causes it to be unable to detect the loss of all fragments of a block at once. Regarding intermediate caching, data is reconstructed at each hop. While this makes sense for dynamic code update (i.e., each node must get all the executable code fragments), that can be very limiting for other applications (audio or image transmission) since it poses significant requirements on node storage capabilities.

3 Distributed Transport for Sensor Networks

The DTSN specification [2] can be divided into a full reliability service, and a differentiated reliability service. The latter is based on the former and is able to support several reliability grades based on the integration of partial buffering at the source, intermediate node caching and erasure coding. This paper focuses on the full reliability service.

The full reliability service was thought for critical data transfer requiring end-to-end full reliability. It is achieved by a Selective Repeat Automatic Repeat reQuest (ARQ) mechanism that uses both negative acknowledgement (NACK) and positive acknowledgement (ACK) packets. Soft requirements of routing path stability and bi-directionality are placed on the routing layer in order to leverage the intermediate node optional cache mechanism of DTSN for performance improvement, although DTSN is able to operate end-to-end as well.

In DTSN, a session is a source/destination relationship univocally identified by the tuple <source address, destination address, application identifier, session number>, designated the session identifier. The session is soft-state by nature both at the source and the destination, being created when the first packet is processed and terminated upon the expiration of an activity timer (provided that no activity is detected and there are no pending delivery confirmations). A randomly chosen session number is appended in order to unambiguously distinguish between successive sessions sharing the endpoint addresses and application identifier. Within a session, packets are sequentially numbered. The Acknowledgement Window (AW) is defined as the

number of packets that the source transmits before generating a confirmation request (Explicit Acknowledgement Request – EAR). The output buffer at the sender works as a sliding window, which can span more than one AW. The size of the output buffer and of the AW depend on the specific scenario, namely on the memory constraints of individual nodes.

The DTSN session management algorithm at the source works as follows. The source transmits each packet coming from the higher layers, storing it in the output buffer, so that it can be retransmitted latter if required. Upon the transmission of each set of packets equal to the AW size, or when the output buffer is full, or when the higher layer protocols have not sent any data during a predefined timeout period, the source requests a delivery confirmation message from the destination by means of an EAR. This may take the form of a bit flag piggybacked in the last data packet (e.g., confirmation request due to the AW size being reached or the output buffer becoming full) or an independent packet (e.g., confirmation request due to the expiration of the EAR timer). Each time a confirmation message (either ACK or NACK) is received, the source frees the output buffer entries whose delivery is confirmed. The reception of an ACK means that there are no gaps in the sequence of packets sent before the respective EAR. On the other hand, a NACK includes a bitmap, where each bit represents a different sequence number (starting from a base sequence number indicated in the packet header) and indicates whether the corresponding packet was correctly received or not. Its reception causes the source to retransmit the data packets that were not delivered successfully. An EAR is sent after retransmission, which may be piggybacked in the last of the retransmitted packets. After sending an EAR, the source launches an EAR timer. If the EAR timer expires before an ACK/NACK is received, the source retransmits the EAR packet.

The DTSN algorithm at the destination works as follows. Upon reception of a data packet with a new session identifier, a new session record is created. However, if a session already exists with the same addresses and application identifier, but the session number is different from the recorded one, the session record is reset and the new session number replaces the old one. The destination then collects the data packets that belong to that flow, delivering in-sequence packets to the higher layer protocol. Upon reception of an EAR, the destination sends an ACK or NACK depending on the existence of gaps in the received data packet stream. Upon the expiration of an activity timer, the session record is deleted and the higher layer protocol is notified in case there are unconfirmed packets.

As already explained, the strict end-to-end transport reliability model used in TCP is not suited for WSNs because it leads to extra consumption of the scarce bandwidth and energy resources. This is caused by the fact that missing packets (and some control packets like NACKs) are retransmitted end-to-end, expending bandwidth and energy in all links/nodes in the path between source and destination. Caching at intermediate nodes is the mechanism employed by DTSN to counter this inefficiency. In DTSN, each node keeps a cache of intercepted packets, managed according to a suitable replacement policy (currently it is FIFO). The packets are stored in cache with probability p, and may belong to different sessions whose end-to-end routing path includes the node in question. Each time an intermediate node receives a NACK packet, it analyses its body and searches for corresponding data packets that are missing at the destination. In case a missing packet is detected, the intermediate node

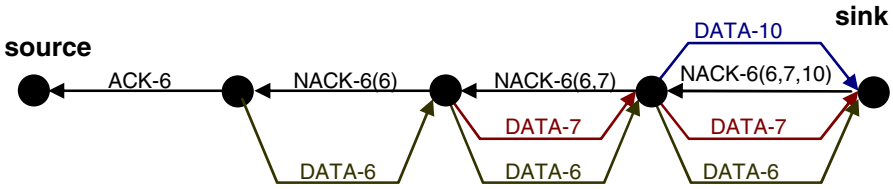

Fig. 1. Example of cache operation and NACK filtering at intermediate nodes

retransmits it. It also changes the NACK contents before resending it, modifying the bitmap so that its retransmitted packets are not included (eventually, the NACK may become an ACK if all missing packets were found in cache). In this way, the source will only have to retransmit those data packets that were not cached at intermediate nodes, decreasing the average hop length of the paths followed by retransmitted packets. Intermediate nodes also discard confirmed packets from the cache and process the session number header field in data packets, replacing any cache entries whose session number is outdated. Fig. 1 illustrates the operation of the cache mechanism for a source-sink path with three intermediate nodes. In this example, the sink node sends a NACK packet stating it is expecting packet 6, so it has received all packets up to packet 5, and packets 6, 7 and 10 are in the missing packet list. The previous node has packet 10, so it sends it to the sink and retransmits the NACK back, removing packet 10 from the missing packets list. The next node has packet 7. The following node has packet 6. As there are no more missing packets, the NACK is turned into an ACK that reaches the source.

The caching probability p at the intermediate nodes should be defined taking into account various factors like cache size, traffic load and EAR frequency. The value of p must be chosen to maximize the probability of cache hit for the NACK requested data along the path. If several nodes have a missing packet cached that is simultaneously retransmitted when they receive a NACK, the resulting collision will degrade performance. To reduce this effect, only nodes in the end-to-end path cache packets in the current DTSN implementation. Ideally, the packets should be cached as close as possible to the destination to reduce the number of radio transmissions necessary to recover from a missing packet. So, the caching probability should decrease with the hop distance to the destination. Naturally, this requires that the routing layer gives this information to the transport layer. To save cache memory, different nodes should cache different packets; consequently the caching probability should be less than 1.

4 Simulation Results

Simulations of the TinyOS 2.x implementation of DTSN were conducted using the TOSSIM simulation environment.

The radiofrequency (RF) parameters of sensor nodes correspond to those of the MICAz motes developed by Crossbow [8], which support a bitrate of 250 kbps based

on the IEEE 802.15.4 standard for the MAC and physical layers [9]. The simulations were made considering a linear topology of evenly spaced sensor nodes, where the transmission power is 0 dBm (maximum power for MICAz nodes), the path loss between each pair of nodes is 70 dB (this corresponds to approximately 10 meters of distance assuming a log-distance path loss model with a distance exponent of 3) and the background noise level is -110 dBm.

Destination-Sequenced Distance-Vector routing (DSDV) [10] is used as the routing protocol underlying DTSN. A continuous traffic pattern was generated at the first node in the line, with the last node as the destination. The data payload size is 2 bytes, adding to the 8 bytes of DTSN overhead, 5 bytes of DSDV overhead and 13 bytes of MAC overhead. The 8 bytes of DTSN overhead include 1 byte for flags (e.g., ACK, NACK), 1 byte for the packet sequence number and 6 bytes for the session identifier. Some overhead may be reduced by combining information of source and destination addresses from the routing layer and transport layers. As a matter of fact, the source and destination addresses could be retrieved from the DSDV routing, saving 4 bytes. Intermediate nodes retransmit cached packets without modification, being transparent to the transport session. An additional byte may be saved by sending the application identifier in the TinyOS Active Message (AM) type field. As these optimizations require cross layer interactions, they were not used in the experiments.

The DTSN transmission window size was configured as 50 packets, with two AWs of 25 packets each. The EAR timeout was set as 250 ms. The maximum number of EAR timeouts is 10. The session activity timeout is 7.5 s at the sender and 10 s at the receiver. The caching probability p (when used) is set to 1 and the cache size is enough for 50 packets. This caching configuration should maximize the cache hit ratio. As a disadvantage, it will also maximize memory consumption. Exhaustive tests with different parameters to save memory were left for future work.

The results allow a comparison between raw DSDV transmission (without reliable transport on top) and DTSN with and without the intermediate node caching mechanism over DSDV routing. Each point in the graphics corresponds to an average over ten independent runs, each consisting of the transmission of 1000 packets back-to-back (subject to local interlayer flow control). The achieved packet loss ratio, user data throughput, average delay and per-packet overhead (measured as the total number of radio transmissions per successfully delivered packet received at the destination) are depicted respectively in Fig. 2, Fig. 3, Fig. 4 and Fig. 5 as a function of the hop distance between source and destination.

The packet loss ratio for DSDV increases quite steadily with the hop distance as noise and unrecovered collisions cause corrupted packets to be lost. This justifies the use of a reliable transport protocol like DTSN that achieves zero packet loss, as expected. On the other hand, DSDV achieves higher throughput and much lower delay as compared with DTSN, which mimics the difference between TCP and UDP in IP networks. The DSDV throughput is naturally higher, since packets have smaller headers. The DTSN delay is much larger than the DSDV delay, as lost packets are retransmitted by DTSN after a timeout. Using an adaptive timeout delay mechanism as used in the TCP protocol may reduce the delay experienced by packets. This investigation was left for further work.

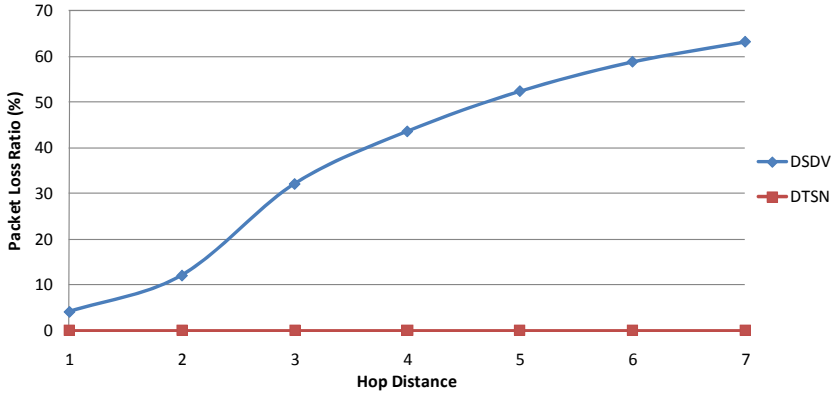

Fig. 2. Packet Loss Ratio

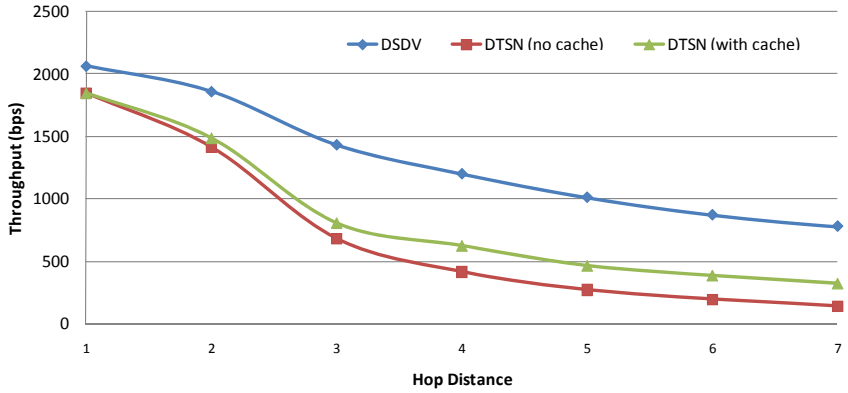

Fig. 3. User data throughput considering 2 data bytes per packet

Regarding the number of radio transmissions per packet shown in Fig. 5, it can be observed that this value increases with the hop distance for all cases. Even though DSDV does not do any retransmission, collisions may trigger MAC layer retransmissions. For this reason, the number of radio transmissions is larger than the hop distance. Although DSDV clearly presents a lower number of radio transmissions as compared with DTSN without cache, the difference is not very significant when compared with DTSN with the cache mechanism turned on. For any hop distance greater than 1, the cache mechanism is used and effectively reduces the number of radio transmissions, saving energy. For hop distances greater than 5, DTSN with cache has even less radio transmissions than DSDV. This result can be justified as fewer packets are effectively delivered in DSDV when the number of hops increases, resulting in increased wasted radio transmissions.

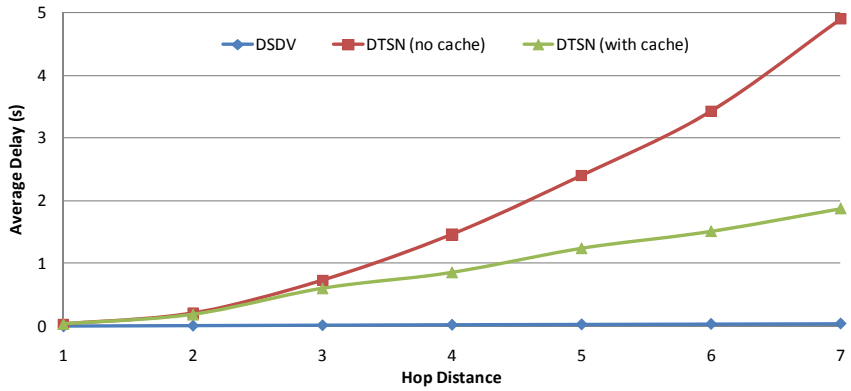

Fig. 4. Average Delay

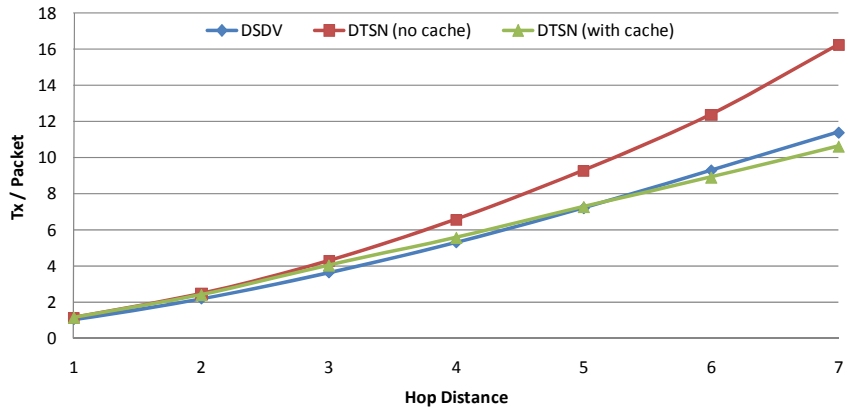

Fig. 5. Average number of RF transmissions per successfully delivered data packet

5 Conclusion

This paper has presented a performance evaluation of the Distributed Transport for Sensor Networks (DTSN). This reliable transport protocol for wireless sensor networks minimizes control overhead by placing the control of the session at the sender, which explicitly requests the data delivery confirmations from the receiver. The use of NACK packets with indication of missing data packets enhances performance. A caching mechanism at the intermediate nodes attempts to minimize end-to-end retransmissions.

The performance evaluation of DTSN was based on a TinyOS 2.x implementation over the DSDV routing protocol. The simulation results attest the effectiveness of DTSN to minimize the number of RF transmissions per packet. The use of the caching mechanism proved to be effective, leveling the energy performance of DTSN

using cache with the performance of the underlying DSDV, with the advantage of having a reliable protocol.

Future work will focus on the evaluation of DTSN for different packet sizes, caching policies, network topologies and traffic load values. Additional evaluations will compare DTSN with other WSN transport protocols and test it with other routing and MAC protocols. Additionally, future work will also include the implementation and evaluation of the differentiated reliability service for Wireless Multimedia Sensor Networks and of an adaptive timeout configuration mechanism.

Acknowledgments. The authors acknowledge the research funding from IST FP6 UbiSec&Sens project and IST FP6 EuroFGI's E2E-WSN specific research project.

References

1. Postel, J.: Transmission Control Protocol, RFC 793, IETF (September 1981)
2. Marchi, B., Grilo, A., Nunes, M.: DTSN – Distributed Transport for Sensor Networks. In: Proceedings of the IEEE Symposium on Computers and Communications (ISCC 2007), Aveiro, Portugal (2007)
3. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System Architecture Directions for Networked Sensors. In: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (November 2000)
4. Stann, F., Heidemann, J.: RMST: Reliable Data Transport in Sensor Networks. In: Proceeding of the 1st IEEE International Workshop on Sensor Net Protocols and Applications, May 2003, pp. 1–11. IEEE, Anchorage, Alaska (2003)
5. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In: Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCON) (August 2000)
6. Dunkels, A., Alonso, J., Voigt, T., Ritter, H.: Distributed TCP Caching for Wireless Sensor Networks. In: Proceedings of the 3rd Annual Mediterranean Ad-Hoc Networks Workshop, Bodrum, Turkey (June 2004)
7. Wan, C.-Y., Campbell, A., Krishnamurthy, L.: PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. In: Proceedings of WSNA 2002, Atlanta, Georgia, USA (2002)
8. MICAz datasheet, Crossbow (24/4/2007), http://www.xbow.com/Products/productdetails.aspx?sid=164
9. IEEE Std. 802.15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) (2003)
10. Perkins, C., Bhagwat, P.: Highly Dynamic Destination-sequenced Distance-vector routing (DSDV) for Mobile Computers. In: SIGCOMM, pp. 234–244 (1994)