A Transport Protocol for Real-time Streaming in Wireless Multimedia Sensor Networks

Duarte Meneses[1], António Grilo[1,2], Paulo Rogério Pereira[1,2]

[1]*Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal*

[2]*INESC-ID/INESC INOV, Rua Alves Redol, nº9, 1000-029 Lisboa, Portugal*

duarte.meneses@ist.utl.pt, antonio.grilo@inov.pt, prbp@inesc.pt

***Abstract*— In this paper, an implementation of a transport protocol to be used in Wireless Multimedia Sensor Networks (WMSNs) is presented. The objective of the protocol is to guarantee reliability by recovering lost packets while dealing with the limited energy, processing and memory capabilities of the devices typically used in these networks. The implemented protocol was Distributed Transport for Sensor Networks (DTSN) and the target device was the embedded device Silex SX-560 using WiFi technology. The implementation was done in C language under the Linux operating system. Following the requirements of the project, additional mechanisms were implemented to enhance the protocol's performance for multimedia data and to provide a certain level of security (authentication and integrity) to the control packets. Results show that the implementation is semantically correct and the protocol is efficient in reducing the total number of transmissions, maintaining a low *overhead*.**

***Keywords*— Wireless Sensor Networks, Multimedia, Transport Protocol.**

I. INTRODUCTION

A Wireless Sensor Network (WSN) is typically formed by a high number of autonomous devices, installed in a region of interest. The nodes cooperate with the objective of collecting significant information about the area. They are equipped with sensors, microprocessors and transceivers, enabling them not only to acquire data, but also to process information and to communicate, creating a wireless network. Sensorial data can be forwarded through many network nodes to a sink, where data is gathered.

Usually the devices are powered by batteries, allowing the easy deployment in any region without increased costs or extra time spent with the installation of a cabled power source, which in some applications might not even be feasible.

With the existence of a variety of different sensors, many physical parameters can be detected or monitored, including: temperature, humidity, light, pressure, soil composition [1]. This set of distributed sensors gives the opportunity of a broad range of new applications in the areas of structure monitoring, home automation, security, military surveillance, target tracking, etc. Many examples can be found in [2] and [3].

Even though WSN are becoming more widespread because of the recent technological advances in microelectronics and wireless communications, usually the devices have very restricting processing power and memory capacity and a finite energy supply.

The joint effort of many nodes in collecting data has many advantages in scalability and introduces innovative applications. However, their characteristics differ from the ones found in traditional computer networks and new communication protocols are required [1]. At the transport layer, research has been focused in congestion control and reliability. Wireless links in WSNs are error prone and thus have a much higher error rate than traditional computer networks. Besides, communication is made across many of those links. Thus, it is important to guarantee reliability in data delivery, implementing loss detection and a retransmission mechanism.

Due to the limited available energy, the transport protocol also needs to be optimized to minimize energy consumption and to extend lifetime, primarily by reducing its overhead and the number of retransmissions. In some cases, energy harvesting techniques that extract energy from the environment can be used to further extend sensor's lifetime, since usually it is not convenient to recharge or change the batteries.

In recent applications, multimedia data such as video, audio or images are acquired using micro-cameras and microphones. Using WSNs, multimedia data can easily be collected from different points or angles. However, multimedia sources generate typically large amounts of real time data, introducing new requirements for WSNs. A higher bit rate is needed to deliver an acceptable multimedia quality and multimedia applications are also usually sensible to delays or jitter. These networks are the wireless multimedia sensor networks (WMSN) [4]. The higher amount of data transmitted and the additional processing needed also turn energy an even more scarce resource in battery powered sensors. It is therefore important to use protocols adapted to the multimedia data transmission, taking advantage of particular characteristics.

One of the applications of WMSNs is the protection of critical infrastructures. That is the objective of Wireless Sensor and Actuator Networks for the Protection of Critical Infrastructures [1] (WSAN4CIP) project, where this work is integrated. The feasibility of such networks will be demonstrated with the application of a WMSN in electrical energy distribution infrastructures. The security and safety measures using WMSN includes sending video through multiple hops from Medium-Voltage/Low-Voltage (MV/LV) electrical power transformers to a High-Voltage/Medium-Voltage (HV/MV) substation. This is one of the specific target applications of the implemented protocol.

The remaining of this paper is organized as follows: a brief overview of WSN reliable transport is presented in section II; In section III, a simple model of the transmission in high loss multi-hop networks is described; section IV gives a general idea about the implementation; finally, the program's test results are shown in section V and conclusions in section VI.

[1] http://www.wsan4cip.eu

II. WSN Reliable Transport

Transport protocols can be categorized according to which problems they address. They may provide some degree of reliability, congestion control or both. We will focus in the reliability issue.

A. *WSN Reliable Transport Protocols*

The purpose is to reliably send data from the source to the final destination. Data is fragmented in multiple packets that are sent and recovered individually. Packets lost in the network have to be recovered through its retransmission. Transport protocols use different feedback mechanisms to notify the source about lost packets, so that it can retransmit them. Most protocols use one or more control packets in the reverse path (in the direction to the data source). Positive acknowledgements (ACK) are sent to notify the source which packets were successfully received, and negative acknowledgements (NACK) are used to inform about lost segments.

The data segments are numbered and losses are usually detected by the receiver by checking the continuity of the segments through their sequence numbers. The source can also detect losses if it does not receive, after some time, the acknowledgment of a packet that it sent.

In WSNs, there are two possible approaches for error recovery: hop-by-hop or end-to-end. Hop-by-hop requires active participation of intermediate nodes. These nodes store data packets in memory so that later they can retransmit them if needed. These nodes can also help detecting packet loss, even though usually their participation is limited to caching and performing retransmissions. With local retransmission, this method is very efficient in reducing retransmission distance, resulting in fewer total transmissions and also even fewer retransmitted packets lost. It is therefore a very important technique in energy constrained WSN as it greatly decreases energy consumption.

In the end-to-end approach, intermediate nodes only forward packets and packet loss detection and recovery has to be made exclusively by the end nodes.

Next, some protocols that implement caching in intermediate nodes are described. More complete surveys about transport protocols for wireless sensor networks can be found in [5], [6] and [7]. DTSN is described in another section in more detail since it is the one implemented.

1) *DTC [8]*

Distributed TCP Caching (DTC) is based on TCP, introducing some optimizations to turn it viable for low resource embedded systems and to improve its performance in WSNs. The advantages of using DTC over other transport protocols specifically designed for WSN is that TCP is widely used in computer networks so WSNs using DTC can be directly connected to an external IP-based network without the need for middle-boxes. The main modifications made to TCP are:

- Header compression: to reduce overhead, which is especially important if the sensor data is just a few bytes per packet;
- Tiny implementation: to be able to run in embedded devices with low memory;
- IP address assignment based on spatial location: to reduce address assignment overhead;
- Distributed cache: to reduce retransmission distance;

2) *RMST [9]*

Reliable Multi-Segment Transport (RMST) was designed to run over the Directed Diffusion [17] network protocol, using its discovered paths, and offers a reliable transport for upstream transmission (from sensor nodes to a sink). The recovery of lost fragments is done using a selective NACK, which is the only control packet used by RMST. The recovery can be made end-to-end or hop-by-hop, depending on a run-time configuration parameter. The sender fragments data into segments and includes in them the fragment number and the last fragment number. With this information, the receiver can detect missing packets. The receiver associates to each missing packet in the sequence a timer, and when it triggers, a selective NACK is sent with the set of the numbers of the fragments that are missing for too long time. When hop-by-hop recovery is used, intermediate nodes cache packets and retransmit them when a NACK with their numbers is received.

3) *PSFQ [10]*

Pump-Slow, Fetch-Quickly (PSFQ) is an upstream transport protocol, offering reliability to multicast transmissions from the sink to many sensor nodes. It basically works with three actions:

- Pump – Data packets are slowly sent from node to node. Each node caches packets so that lost packets recovery is made hop-by-hop. Each node waits a random time before forwarding the packets. The objective of this delay is to avoid collisions, avoid unnecessary redundant transmissions and to allow retransmission of packets even before the next segment is received.
- Fetch – Any node enters in fetch mode when it detects a jump in the segment sequence. In some cases, fetching can also proactively occur based on timeouts. A NACK is sent to the neighbours, informing them about missing segments. Each neighbour might send different segments according to what they have in cache, waiting once again a random time interval.
- Report – The sink can include a bit in the data packets explicitly requesting a report about data delivery. Nodes answer back in unicast to the sink. Intermediate nodes might aggregate multiple reports while forwarding them to the sink.

B. *DTSN overview*

DTSN [11] is a reliable generic upstream transport protocol designed specifically for WSNs. Its objective is to provide reliable data delivery while minimizing the energy consumption. Data is transmitted in the context of sessions that are explicitly created by the application. These sessions are internally implicitly created and are univocally identified by a session number, an application id, the source address and the destination address. Every packet carries this identification in its header, along with a sequence number that orders the packet in the sequence, allowing the reception of out of order packets. Recovery of lost packets is achieved by a selective repeat mechanism. DTSN uses both accumulative ACKs and selective NACKs which also act as an accumulative ACK since they include the sequence number of the last packet successfully received in order. NACKs also include a bitmap with the list of missing packets in the receiver window. These packets are sent by the receiver only when explicitly requested by the source, through an explicit acknowledgment request

(EAR). This request is a bit that is piggy-backed in data packets or sent as a separate control packet. The EAR is sent in many situations, for example, after a certain number of new packets have been sent. A timer is used to detect the loss of an EAR or its answer, triggering another request.

Even though only the source and the destination create control packets, the intermediate nodes also participate in the recovery of lost packets. These nodes cache data packets that cross them, so that later, if needed, they can retransmit them. To accomplish this, they intercept NACK packets and retransmit the packets requested in its bitmap that are found in the cache. The bitmap of the NACK is modified before forwarding it, so that the packets sent are not requested anymore. In this way, the packets can be retransmitted from a point closer to the link where it was previously lost, reducing the retransmission distance and saving energy.

III. Analytical Model of WSN Transport Caching

In this section, a simple model of the transmission in high loss multi-hop networks, from the transport layer perspective, is used. Lower layers in the network stack can do, up to some extension, local retransmissions through *Automatic Repeat reQuest* (ARQ) mechanisms. It is assumed that a certain level of reliability is offered to the transport layer, expressed as a probability of bit transmission success in each link. The impact of bit errors on the number of control packets transmitted is ignored. This is reasonable to do since the data volume of these packets is much lower than the data volume of the data packets.

The probability of end-to-end success, S_e, in a network with H hops and with a probability P_l of losing a packet in each link, is given by the following expression:

$$S_e = (1 - P_l)^H \tag{1}$$

Figure 1 represents the end-to-end success probability for networks with various numbers of hops and link loss probability. Even though in each link the losses are relatively low, in networks with many hops the end-to-end situation is much worse, with success probability under 50% in many situations.

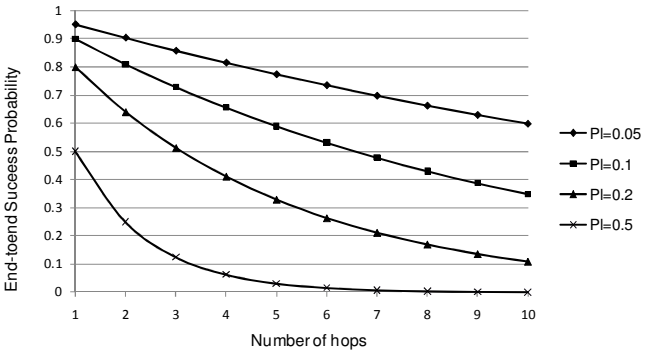

Figure 1 – End-to-end success probability versus number of hops and link-level loss probability.

We would also like to know how many links, in average, a packet crosses before being lost or successfully received by the destination node. This value can be calculated with:

$$L = H.(1 - P_l)^H + \sum_{h=1}^{H-1} P_l.(1 - P_l)^h.h \tag{2}$$

With the additional information that the packet was lost, the expression changes to:

$$L_{lost} = \frac{\sum_{h=1}^{H-1} P_l.(1 - P_l)^h.h}{1 - S_e} = \frac{\sum_{h=1}^{H-1} P_L.(1 - P_l)^h.h}{1 - (1 - P_l)^H} \tag{3}$$

Table I shows some values of the expected number of links traversed by the packets and the corresponding fraction of the total path.

TABLE I
AVERAGE NUMBER OF LINKS CROSSED BY EACH DATA PACKET

P_l \ hops	2	4	6	8	10
0.01	1.97 (98.5%)	3.90 (97.5%)	5.75 (96.6%)	7.65 (95.6%)	9.47 (94.7%)
0.05	1.85 (92.7%)	3.52 (88.1%)	5.03 (83.9%)	6.40 (79.9%)	7.62 (76.2%)
0.10	1.71 (85.5%)	3.10 (77.4%)	4.22 (70.3%)	5.13 (64.1%)	5.86 (58.6%)
0.15	1.57 (78.7%)	2.71 (67.7%)	3.53 (58.8%)	4.12 (51.5%)	4.55 (45.5%)
0.20	1.44 (72.0%)	2.36 (59.0%)	2.95 (49.2%)	3.33 (41.6%)	3.57 (35.7%)

When caching is not used, the expected number of link-level transmissions needed to successfully send a packet to the destination can be calculated with L_{lost}:

$$Tx_{nocache}(H) = H + (L_{lost} + 1).\sum_{i=1}^{\infty}(1 - S_e)^i \tag{4}$$

In the situation in which caching is enabled in intermediate nodes and assuming that they have capacity to store all the packets needed for local retransmission, we can model the transmission over H hops as H independent transmissions over 1 hop. Thus, the expected number of transmissions is given by:

$$Tx_{cache}(H) = H.\sum_{i=0}^{\infty}(i + 1).(1 - P_l).{P_l}^i = H.\sum_{i=0}^{\infty}{P_l}^i \tag{5}$$

Figure 2 represents the data packets' transmission with graphs: a) when caching is disabled and b) when caching is enabled. The transitions in red represent transmissions.

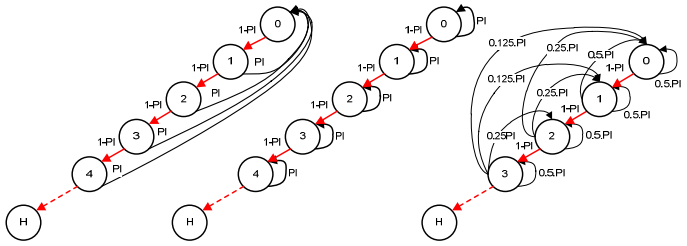

Figure 2 –Graph representing the probabilistic model of data packet transmission. a) caching disabled b) caching enabled c) caching with 50% of probability.

In practice, losses near the receiver can be aggravated by congestion, if there is an aggregation of multiple links. In this case, storing packets in middle nodes can be even more important. To assess the advantages, we consider also a situation in which the last link, next to the receiver, has a packet loss probability of 0.25. To calculate the new values, we generalize formulas (3), (4) and (5) to allow variable losses in each link:

$$L_{lost,g} = \frac{\sum_{h=1}^{H} P_l^{(h)}.\left(1 - P_l^{(h)}\right)^{h-1}.h}{1 - \prod_{i=1}^{H}(1 - P_l^{(h)})} \tag{6}$$

$$Tx_{cache,g}(H) = \sum_{h=1}^{H} \sum_{i=0}^{\infty} P_l^{(h)^i} \quad (7)$$

$$Tx_{nocache,g}(H) = H + (L_{perd,g} + 1).\sum_{i=1}^{\infty}\left(1 - \prod_{h=1}^{H} P_l^{(h)}\right) \quad (8)$$

$P_l^{(h)}$ is the packet loss probability in link h.

Often, nodes have limited memory available and thus cannot store all the packets from the sliding windows of all active sessions to which the nodes belong to the path. To assess the performance of the hop-by-hop recovery mechanism using caching in these conditions, we consider now that packets are cached with 50% probability. This is a possible caching policy to be used in these systems. In this situation, retransmission can be made from any of the nodes in the forward path that are behind the link where the loss occurred, with different probabilities, as illustrated in Figure 2 c). Using (4), we can calculate the expected number of transmissions for this case, with the following formula:

$$Tx_{probcache}(H) = H + \sum_{i=1}^{\infty}(P_l^{\,i}).\sum_{h=1}^{H}\Bigg[1 + 0.5^{h-1}.Tx_{nc}(h-1) + \sum_{j=2}^{h-1} 0.5^{h-j}.Tx_{nc}(h-j)\Bigg] \quad (9)$$

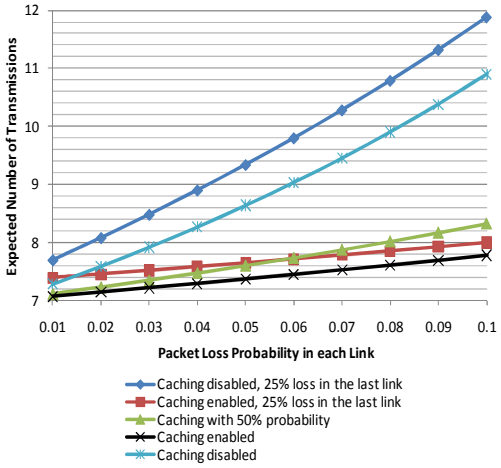

Figure 3– Expected number of transmissions per packet, in a network with 7 hops and variable packet loss probability in each link, for 5 different situations.

Values were calculated for all the 5 described situations, and are represented in Figure 3.

Using cache and for the used parameters, the additional transmissions used to recover lost packets is in average below one. Only one extra transmission needs to be made as a consequence of a packet loss, wherever it occurs.

Without the participation of the middle nodes in the recovery of lost packets, packet loss results in a much higher number of additional transmissions, which increases with a much higher slope with the increase of the packet loss probability.

The results show that storing packets with only 50% probability is still efficient in reducing the average number of transmissions, because the retransmission distance is greatly shortened. It is a great improvement from the situation with no cache at all and could be a good solution for nodes with limited memory.

The size of the payload of data packets is important as the efficiency of the transmission depends on it. The larger the packet is, the larger is the probability of losing the packet. Also, if an error occurs, the packet has to be retransmitted, which costs more the larger the packet is. On the other hand, small data packets have the disadvantage of increasing the protocol's overhead.

In DTSN, when security is enabled, the overhead in data packets is 51 bytes. Figure 4 shows the total amount of bytes transmitted while transferring 2×10^6 bytes of data through a 7 hop network, with a bit error probability of 2.5×10^{-5} in each link and 100% caching probability. The DTSN overhead was considered in the calculations and once again we ignored the control packets traffic. For small payloads, the headers overhead can be observed, while for larger payloads, the packet errors impact is noticeable. It can be observed that the performance is much less dependent on the payload size when the cache is used.

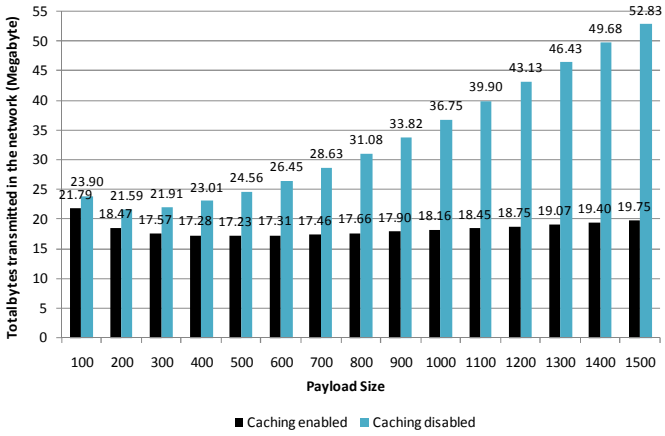

Figure 4 – Total amount of data transmitted in a 7 hop network, with a bit error probability of 2.5E-5.

IV. DTSN Implementation

A. *Target Platform and OS*

DTSN is a generic protocol and can be implemented in various platforms using different wireless networks. A comparison of the characteristics of some wireless technologies typically used in WSNs can be found in [12] and [13]. The chosen technology for this particular implementation was WiFi, as it best suits the requirements of most multimedia applications. WiFi offers adequate range and bit rate and the embedded devices that use it have typically sufficient processing capacities and satisfactory power consumption for most applications in WMSN.

The target device was the Silex SX-560 [14]. It is an embedded device and uses WiFi technology, supporting the IEEE standards 802.11a/b/g. It is suitable for battery powered applications since it requires a 3.3V energy supply and has an average current consumption of 240 mA. The hardware comprises a 32 bit ARM processor with a 200 MHz clock, 16 MB RAM and an 8 MB flash disk. These processing and memory capabilities are adequate for the required processing and for caching of packets in intermediate nodes.

The device runs a light version of the Linux operating system, making it easy to develop applications in the C programming language. The used C library, uClibc, is much smaller than the traditional glibc and was developed

specifically for embedded systems. It supports all POSIX functions (including threads) and shared libraries, which turned out to be useful for this implementation.

B. *Software Architecture*

The core of the protocol runs in a daemon process (Figure 5). This process implements all the functionalities of the protocol since all the nodes – source, receiver and intermediate ones - run the same process. The communication between nodes is done through UDP sockets. The daemon sets up, on start up, a UDP port were all packet exchange is made through. UDP was used to simplify implementation and debugging. In a future version, raw sockets may be used to save UDP overhead.

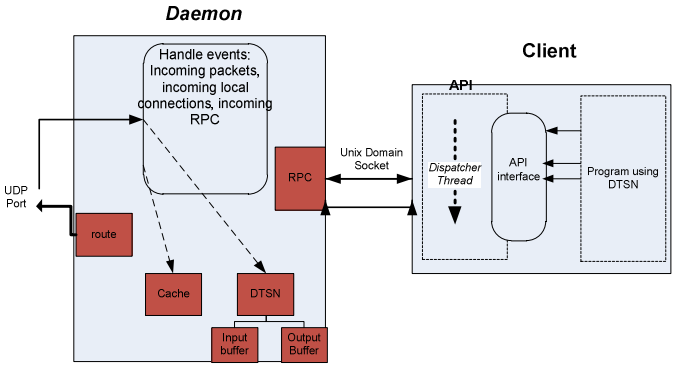

Figure 5 – DTSN software architecture

An API was developed to create a simple interface for applications that want to use DTSN. This interface is similar to the ones usually found in Linux sockets. For example, many functions implement a blocking functionality which can be arbitrarily time limited. The application can also optionally register a callback function, where events are notified.

The API is a shared library that programs can load. This library communicates with the daemon through two UNIX sockets: one for remote procedure calls (RPC) and another for transmission of asynchronous events to the application. This model requires having a thread continuously waiting for events, to transmit them to the application and to release blocked threads based on them (Figure 6).

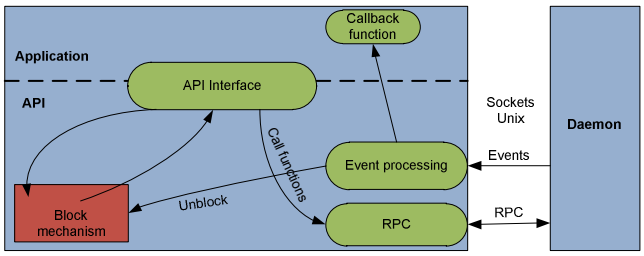

Figure 6 – API operation

The daemon is constituted by a single thread, running synchronously. It has to handle incoming packets, incoming local connections, incoming RPC requests and timeouts. The sending and receiving windows of sessions are mapped to circular buffers. Both the sender and the receiver have the windows synchronized through the control packets. NACK and ACK packets allow the sender's window to advance and the EAR packet specifies the last sent packet and might correct the receiver's window. This mechanism implements a simple flow control. Also a simple routing module was developed for testing purposes, to simulate packet loss and to perform arbitrary packet routing not dependent on the IP addresses. Many protocol's parameters, as timeouts, cache sizes and windows sizes are configurable through a configuration file that is loaded on startup.

C. *Security*

As most transport protocols for WSN, DTSN does not consider security, usually assuming that lower layers will provide security if needed. However, this can be energetically inefficient and require overhead, both in network traffic and in processing time and the intermediate nodes would need to be included since they have to be allowed to manipulate DTSN packets. Instead, an extension to DTSN was designed to ensure authentication and integrity of the control packets [15]. The objective is to protect the network against packet injection, modification or repetition attacks by providing authentication. It uses a cryptographic symmetric function: a hash-based message authentication code based on the SHA-2 algorithm. Based on a secret password that only the source and destination possess and on the session number, the source and destination create different ACK and NACK keys for every different sequence number.

ACK packets include the ACK key corresponding to its sequence number. The data source can verify its authenticity generating the key for the ACK sequence number and comparing the keys. NACK packets also include the ACK key since they also have the acknowledgment function, and include the NACK key for every sequence number requested in the bitmap. Each data packet includes a message integrity code (MIC). MICs are calculated with the same hash function as the keys, and use as parameters the packet and the NACK key that corresponds to the data packet's sequence number. The receiver can verify if the packet is legitimate by recalculating the MIC using its own generated key.

The intermediate nodes cache data packets with the MIC. When a NACK is received, it only retransmits a data packet if it can successfully calculate the same MIC with the corresponding NACK key included in the NACK packet.

D. *Video Transmission*

In one of the protocol's target application, video is transmitted from micro-cameras to a sink. Some encoders use scalable video coding techniques, in which each video frame has multiple layers. A base layer contains the critical information and is enough to render the picture with a minimum quality while the other layers are enhancement layers that provide additional quality to the picture. Focusing in this particular kind of data, transmissions mechanisms were designed to improve the efficiency of the transmission of such multimedia data with DTSN. This modified version of the protocol is the multimedia DTSN (M-DTSN) [16], which was included in this Linux implementation.

A function was added to the interface that allows the application to reset a session. Resetting a session causes the source to drop all data packets in the sending window and start sending new data packets with a new ID. When receiving the packets with the new ID, the intermediate nodes clear the session's cache and the receiving node also discards old packets in the receiving window. The objective of this technique is to limit each frame's sending time. After that time, all efforts to transmit the unconfirmed frames are dropped and a new frame immediately starts to be transmitted. This is important for real-time streaming.

V. Performance Evaluation

The implementation was tested in a LAN network. A routing module was added to the program, which routes packets through 7 hops and simulates packet loss in each link, based on a bit error probability. A 1.95 MB file was transferred in this simulated WSN and statistics were saved in each node, allowing us to draw many conclusions about the protocol's performance.

A. *Caching enabled*

We start by showing the results from the tests performed with caching enabled and security disabled.

Figure 7 shows the average data packet transmissions made by all the nodes to send each data segment from the source to the final destination. It also compares the values acquired in the test with those calculated using the proposed model. Notice that the results are very close (maximum error 1.4%).

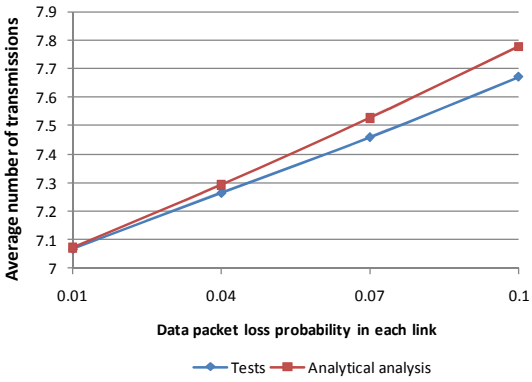

Figure 7 – Average number of transmitted data packets per segment, with caching enabled

Figure 8 shows the impact of the packet loss and of the protocol overhead in the transmitted data volume. These values, especially the protocol overhead, are strongly dependent on the data packets' payload. With the used payload (512 bytes) DTSN overhead is very reasonable, staying below 5%. It increases slightly when the loss probability increases because the number of control packets transmitted also increases.

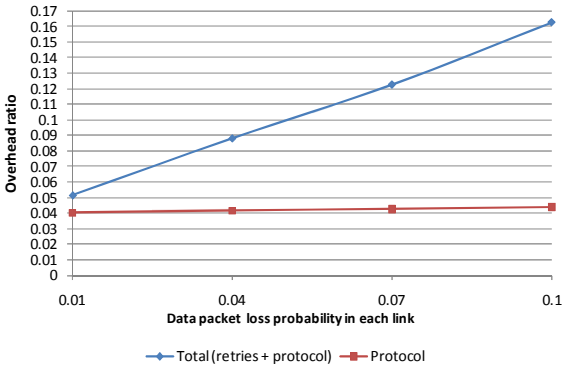

Figure 8 – Overhead ratio with caching enabled

B. *Caching disabled*

The same tests were repeated, but this time with caching disabled in the intermediate nodes.

Figure 9 illustrates the average number of data packets transmissions per sent segment and the average number of traversed links by the data packets before being lost or successfully received by the destination, acquired both through the model and through the tests. Both results are, once again, very close.

In Figure 10 we can observe more details regarding the data packets lost in each link. Unlike the case where caching is enabled, nodes closer to the source are crossed by more packets, and as a result we can observe a higher packet loss in links nearer the source.

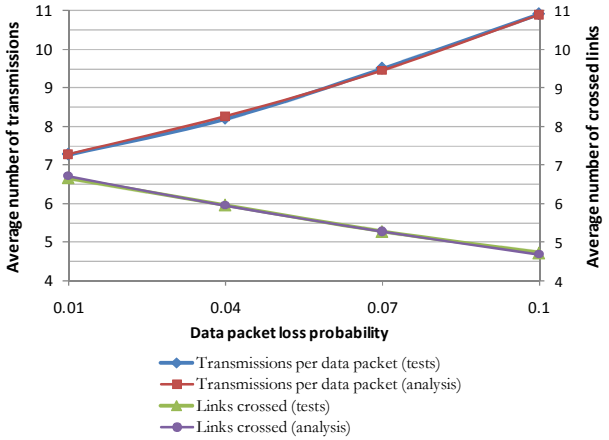

Figure 9 –Average number of transmissions per segment and average number of links crossed by each data packet, when caching is disabled.

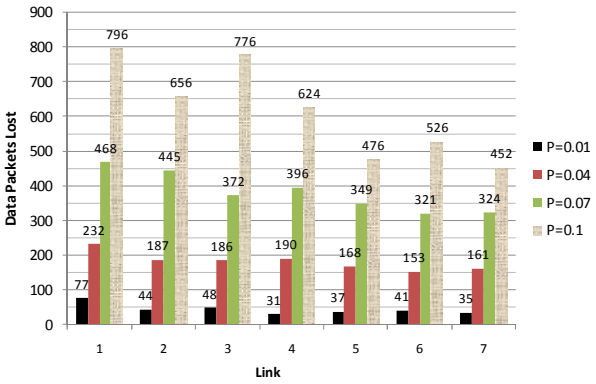

Figure 10 - Data packets lost in each link, with caching disabled

C. *Security enabled*

The same file transfer test was performed with security and caching enabled. The bit error probabilities simulated were the same as in the last sections, but this time three different payloads were used (200, 350 and 512 bytes) to better understand its effect in the protocol's performance.

Figure 11 shows the total data packet loss for the 5 different tested situations, normalized by the number of segments sent (tests with smaller payloads have more segments). The smaller the payload is, the smaller is the number of losses per segment, since the probability of losing each packet decreases. Note that with security there are more losses as the packets are larger. Without caching, there is also more packet loss because the retransmitting distance is larger, increasing the number of links where the retransmitted packet can be lost. Therefore, without caching, not only it is energetically more expensive to retransmit packets, but there are also more packets lost.

Figure 12 compares the average number of necessary data packet transmissions to successfully send a segment to the final destination. The use of caching greatly reduces transmissions, saving energy.

Figure 13 shows the total data volume for all 5 tested situations. This is the full amount of data transmitted,

including control packets. Essentially, this is the value we are interested in minimizing to reduce energy consumption. Enabling security, not only increases the packet size, but also, for the same reason, turns packet loss more likely. For low error probability, using security has lower performance than the situation with cache and security disabled. With cache, the total bytes transmitted are much less dependent on the networks conditions. From the 3 situations with cache and security enabled, the most efficient way to transmit data is using 512 bytes payload. This result matches the one obtained from the analysis.

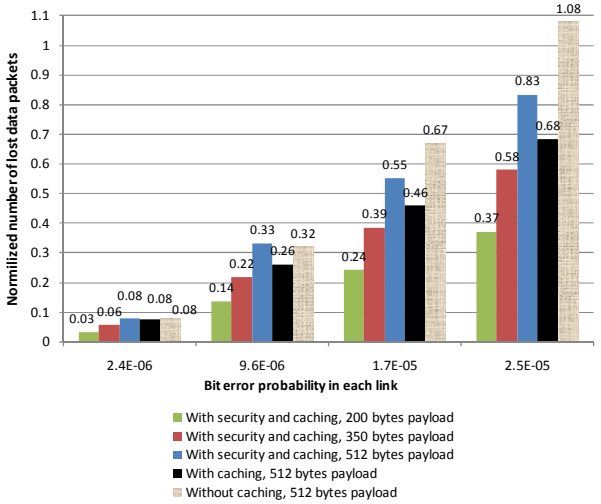

Figure 11- Number of data packets lost, normalized with the total number of segments sent

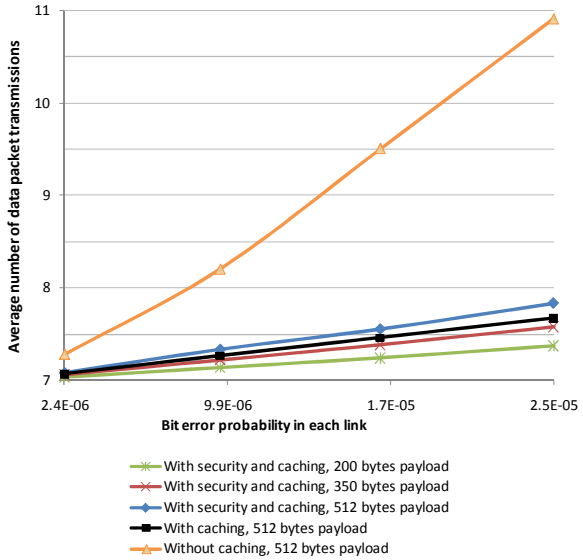

Figure 12 – Average number of data packet transmission per sent segment, in the five tests.

Data volume corresponding to control packets is relatively small, as it can be observed in Figure 14. On the other hand, when security is enabled, the total protocol overhead is high. Figure 15 shows the overhead ratio for the tested situations. Once again, the values are highly dependent on the data packets' payload size.

D. *Maximum Data Rate*

Finally, the data transfer rate between a PC and a Silex SX-560 was tested, using the implemented transport protocol. Both computers were connected through a 1 hop Ethernet connection and with the appropriate configuration, there is no buffer overflow and hence no packet loss occurs. In this situation, the maximum data rate is limited by the processing capacity of the Silex SX-560. A 1.95MB file was used for these tests.

Table II compares the performance of the implemented DTSN with FTP and TFTP in transferring the file in both ways.

Despite TCP having better transfer times in these tests, they do not reflect the performance in high loss multihop networks with large delays as WSNs, where the selective retransmission and caching bring a significant advantage. The results shown depend mainly on the processing speed.

Benchmarking DTSN, we found out that the bottleneck is the communication between the API and daemon, which creates a considerable delay because of the use of a mutex to guarantee exclusive access, synchronization and data transfers. To confirm this, in Table II is also included the data rate achieved when no data is delivered to the API. As expected, security does affect the data rate, however the performance hit is very low.

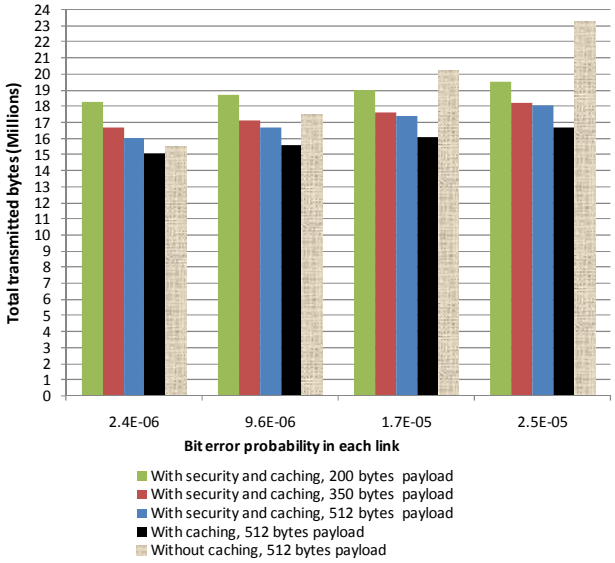

Figure 13 – Total transmitted bytes in all computers, during the five tests.

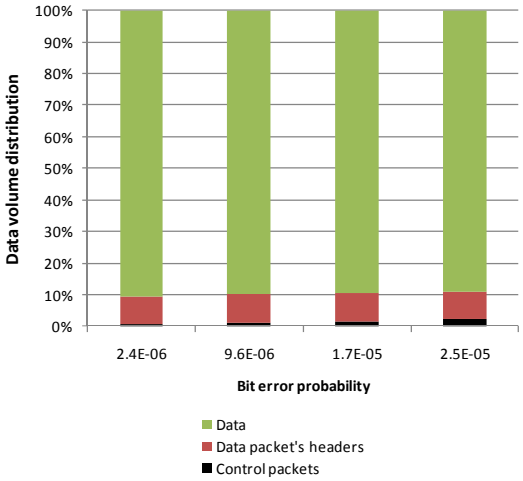

Figure 14 – Data volume distribution, with security enabled and 512 bytes payload.

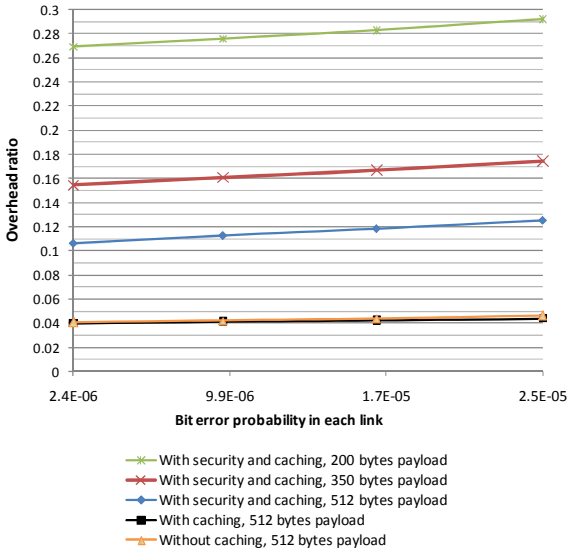

Figure 15 – DTSN overhead ratio in the five tests

TABLE II
DATA RATES AND TRANSFER TIMES

	PC → Silex	**Silex → PC**
DTSN	10.36 s (1.51 Mbps)	8.03 s (1.94 Mbps)
DTSN with security	10.97 s (1.42 Mbps)	8.22 s (1.90 Mbps)
DTSN without delivery to application	4.69 s (3.33 Mbps)	8.03 s (1.94 Mbps)
TFTP (over UDP)	12.0 s (1.30 Mbps)	79.9 s (0.20 Mbps)
FTP (over TCP)	3.0 s (5.20 Mbps)	5.17 s (3.02 Mbps)

VI. CONCLUSION

In this paper, we presented an implementation of a transport protocol intended to provide reliable delivery of multimedia data in wireless sensor networks.

A simple analytical model of multihop networks was proposed, allowing to easily draw some conclusions about the characteristics of data transmission in a multihop networks.

A transport protocol specifically designed for WSN, DTSN, was successfully implemented in an embedded device using WiFi. Additional features were also implemented to improve security and the performance with multimedia data and to provide authentication to the control packets. The program has a simple interface for applications through a shared library. The implementation was tested by simulating a WSN in a LAN.

The values acquired through analysis and the practical test results match and both show that the caching mechanism is a very important feature in WSNs to reduce the total amount of bytes transmitted in the network and therefore reduce the global consumed energy. By reducing the retransmission distance, it also reduces packet delivery delay.

Some possible future work topics are implementing adaptive timeout mechanisms, congestion control mechanisms and other caching mechanisms.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement no. 225186. Consortium: Eurescom, IHP Microelectronics, NEC Europe, INOV, EDP Distribuição, Budapest University of Technology and Economics, INRIA, Lulea University of Technology, Sirrix, Tecnatom, University of Malaga and FWA. The research also received funding from FCT (INESC-ID multiannual funding) through the PIDDAC program funds.

REFERENCES

[1] Zheng, Jun; Abbas, Jamalipour; *Wireless Sensor Networks – A Networking Perspective*. Hoboken, New Jersey: Wiley, 2009.

[2] Yick, J.; Mukherjee, B.; Ghosal, D.; "Wireless sensor network survey", *Computer Networks 52:12*, pp. 2292-2330, August 2008.

[3] Akyildiz, Ian F.; Melodia, Tommaso; Chowdhury, Kaushik R.; "A survey on wireless multimedia sensor networks", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, v.51 n.4, p.921-960, March, 2007.

[4] Almeida, J.; Grilo, A.; Pereira, P. R.; "Multimedia Data Transport for Wireless Sensor Networks" *Next Generation Internet Networks, 2009. NGI'09*, pp.1-8, 1-3 July 2009.

[5] Rathnayaka, A.J.D.; Potdar, V.M.; Sharif, A.; Sarencheh, S.; Kuruppu, S.; "Wireless Sensor Network Transport Protocol - A State of the Art", *International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA'2010)*, pp.812-817, 4-6 Nov. 2010.

[6] Chonggang Wang; Sohraby, K.; Bo Li; Daneshmand, M.; Yueming Hu; "A survey of transport protocols for wireless sensor networks", *IEEE Network*, vol.20, no.3, pp. 34-40, May-June 2006.

[7] Wang, C.; Sohraby, K.; Li, B.; Tang, W; "Issues of transport control protocols for wireless sensor networks", *International Conference on Communications, Circuits and Systems,* vol.1, pp. 422-426, 27-30 May 2005.

[8] Voigt, Thiemo; Dunkels, Adam; Alonso, Juan; "Reliability in distributed TCP caching". *Workshop on Sensor Networks Workshop at Informatik 2004*, Ulm, Germany, September 2004.

[9] Stann, F.; Heidemann, J.; "RMST: reliable data transport in sensor networks", *First IEEE International Workshop on Sensor Network Protocols and Applications,* pp. 102-112, 11 May 2003

[10] Wan, C.Y.; Campbell, A.T.; Krishnamurthy, L.; "Pump-slowly, fetch-quickly (PSFQ): a reliable transport protocol for sensor networks", *IEEE Journal on Selected Areas in Communications,* vol. 23, no. 4, pp. 862-872, April 2005.

[11] Rocha, F.; Grilo, A.; Pereira, P.; "Performance Evaluation of DTSN in Wireless Sensor Networks", *4th EuroNGI Workshop on Wireless and Mobility*, Barcelona, Spain, January 2008. In Springer-Verlag Lecture Notes in Computer Science, vol. 5122, 2008.

[12] Lee, Jin-Shyan; Su, Yu-Wei; Shen, Chung-Chou; "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi", *33rd Annual Conference of the IEEE Industrial Electronics Society, IECON 2007*, pp.46-51, 5-8 Nov. 2007.

[13] Boudhir; Bouhorma; Benslimane. *"Quality of Service and Communication Technologies for Wireless Multimedia Sensor Networks"*. 17th Telecommunications forum TELFOR, Belgrade, Serbia. 2009.

[14] Silex Technology, "SX-560 Intelligent Programmable WLAN Module", SX-5602009EN, Internet: http://www.silexeurope.com/media/datasheets/SX-560-ds_en_0902.pdf, 2009.

[15] Buttyan, L.; Grilo, A. M.; "A Secure Distributed Transport Protocol for Wireless Sensor Networks", *IEEE ICC 2011*, Kyoto, Japan, June 2011.

[16] Mingorance-Puga, J.F.; Macia-Fernandez, G.; Grilo, A.; Tiglao, N.M.C.; "Efficient multimedia transmission in wireless sensor networks", *6th EURO-NF Conference on Next Generation Internet (NGI'2010)*, pp.1-8, 2-4 June 2010.

[17] Intanagonwiwat, C.; Govindan, R.; Estrin, D.; Heidemann, J.; Silva, F.; "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 2-16, Feb 2003.