A Secure Distributed Transport Protocol for Wireless Sensor Networks

Levente Buttyan
Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics, Hungary
http://www.crysys.hu/

António M. Grilo
INESC-ID, Instituto Superior Técnico
Technical University of Lisbon, Portugal
http://comp.ist.utl.pt/

***Abstract*—We propose a secure distributed transport protocol for wireless sensor networks that resists against attacks on the reliability service provided by the protocol, as well as against energy depleting attacks. Our protocol is based on the Distributed Transport for Sensor Networks (DTSN) protocol, to which we add a security extension that consists in an efficient, symmetric key based authentication scheme for control packets. Besides describing the operation of our protocol, we also provide its analysis in terms of security and overhead.**

I. Introduction

In some applications of Wireless Sensor Networks (WSN), for instance, in case of multimedia sensor networks [1], the sensors capture and transmit high-rate data with some QoS requirements. Such applications require the use of a transport protocol that ensures reliable delivery and congestion control. It is widely accepted that transport protocols used in wired networks (e.g., the well-known TCP) are not applicable in WSNs, because they perform poorly in a wireless environment and they are not optimized for energy consumption. Therefore, a number of transport protocols specifically designed for WSNs have been proposed in the literature (see e.g., [2] for a survey). The main design criteria that those transport protocols try to meet are reliability and energy efficiency. However, despite the fact that WSNs are often envisioned to operate in hostile environments, they do not address security issues at all, and as a consequence, they ensure reliability and energy efficiency only in a benign environment where no intentional attack takes place [3].

Attacks against WSN transport protocols come in two flavors: attacks against reliability and energy depleting attacks. An attack against reliability is considered to be successful if the loss of a packet (or packet fragment) remains undetected. In case of energy depleting attacks, the goal of the attacker is to force the sensor nodes to perform energy intensive operations, in order to deplete their batteries. An example would be when the attacker coerces some sensor nodes to unnecessarily re-transmit packets (or packet fragments).

In this paper, we propose a secure WSN transport protocol, which, to the best of our knowledge, is the first of its kind. Our protocol is based on the Distributed Transport for Sensor Networks (DTSN) protocol [4], to which we propose a security extension that transforms the otherwise vulnerable DTSN protocol into a secure WSN transport protocol that resists both reliability and energy depleting attacks. Essentially, our proposal consists in an authentication scheme that prevents the forgery of transport control packets, i.e., acknowledgements (ACK) and negative acknowledgements (NACK). The prevention of ACK forgery ensures that the attacker cannot generate a valid ACK for a valid packet that has not been received by the destination, and hence, the loss of a valid packet can be detected by not receiving an ACK for it. The prevention of NACK forgery ensures that the attacker cannot inject valid NACKs that would trigger the retransmission of the corresponding valid data packets. As a side effect, our authentication scheme also provides end-to-end authentication and integrity protection of the data packets.

The organization of the paper is the following: In Section II, we briefly summarize the operation of the DTSN protocol. In Section III, we analyze the security vulnerabilities of DTSN and define our design objectives. In Section IV, we describe our proposed authentication scheme, and we analyze its security and overhead in Section V. Finally, we conclude the paper in Section VI.

II. The DTSN protocol

DTSN [4] is a reliable transport protocol developed for sensor networks where intermediate nodes between the source and the destination of a data flow cache data packets in a probabilistic manner such that they can re-transmit them upon request. The advantage of allowing intermediate nodes to cache and re-transmit data packets is that the average number of hops a re-transmitted packet must travel is smaller than the length of the route between the source and the destination. In case of a fully end-to-end reliability mechanism, where only the source is allowed to re-transmit lost data packets, re-transmitted packets always travel through the entire route from the source to the destination. Thus, DTSN improves the energy efficiency of the network compared to a transport protocol that uses a fully end-to-end re-transmission mechanism. Our proposed security extension that we will introduce later preserves this advantageous feature of DTSN.

DTSN uses special packets to control caching and re-transmissions. More specifically, there are three types of such control packets: Explicit Acknowledgement Requests (EARs), Positive Acknowledgements (ACKs), and Negative Acknowledgements (NACKs). The source sends an EAR packet after

the transmission of a certain number of data packets, or when its output buffer becomes full, or when the application has not requested the transmission of any data during a predefined timeout period. Upon the reception of an EAR packet, the destination sends an ACK or a NACK packet depending on the existence of gaps in the received data packet stream. An ACK refers to a packet sequence number n, and it should be interpreted such that all packets with sequence number smaller than or equal to n were received by the destination. A NACK refers to a base sequence number n and it also contains a bitmap, in which each bit represents a different sequence number starting from the base sequence number n. A NACK should be interpreted such that all packets with sequence number smaller than or equal to n were received by the destination and the packets corresponding to the set bits in the bitmap are missing.

Within a session, packets are sequentially numbered. The Acknowledgement Window (AW) is defined as the number of packets that the source transmits before generating an EAR. The output buffer at the sender works as a sliding window, which can span more than one AW. Its size depends on the specific scenario, namely on the memory constraints of individual nodes.

In DTSN, besides the source, intermediate nodes process ACK and NACK packets too. When an ACK packet with sequence number n is received by an intermediate node, it deletes all data packets with sequence number smaller than or equal to n from its cache and passes the ACK packet on to the next node on the route towards the source. When a NACK packet with base sequence number n is received by an intermediate node, it deletes all data packets with sequence number smaller than or equal to n from its cache, and in addition, it re-transmits those missing data packets that are indicated in the NACK packet and stored in the cache of the intermediate node. The bits that correspond to the re-transmitted data packets are cleared in the NACK packet, which is then passed on to the next node on the route towards the source. If all bits are cleared in the NACK, then it essentially becomes an ACK referring to the base sequence number. The source manages its cache and re-transmissions in the same way as the intermediate nodes, without passing on any ACK and NACK packets.

III. Security issues and design objectives

As we have shown earlier in [3], reliable transport protocols are vulnerable to control packet manipulation attacks mounted by compromised nodes on the route between the source and the destination. In particular, a compromised intermediate node can forge or alter an ACK packet on its way to the source, which creates the false impression that data packets have been received by the destination while in reality they may have been lost. Data packets that are believed to be delivered are deleted from the caches of intermediate nodes and the output buffer of the source. Thus, forging or altering ACK packets may lead to permanent loss of some data packets, and puts the reliability service provided by the protocol in jeopardy. A compromised intermediate node can also forge or alter NACK packets, which would have a similar effect, as a NACK also acknowledges the reception of data packets up to the base sequence number in the NACK. In addition, setting bits in the bitmap in a NACK packet triggers unnecessary re-transmission of the corresponding data packets, leading to increased energy consumption and a larger probability of packet loss due to increased interference. While unnecessary re-transmissions do not directly harm the reliability service provided by the DTSN protocol, it is clear that such inefficiency is still undesirable.

ACK and NACK packets can also be entirely deleted by compromised intermediate nodes, which cannot be prevented by cryptographic countermeasures. Note, however, that an eventually received later ACK or NACK packet with a larger sequence number or base sequence number, respectively, acknowledges all previous data packets including those referred to by the deleted ACK or NACK, and hence, ensures recovery from their loss. The deletion of NACK packets, or just clearing set bits in their bitmap, will also prevent the immediate re-transmission of the corresponding missing data packets, but again, the eventual reception of an intact NACK packet ensures recovery from this situation too.

Moreover, a compromised intermediate node may replay previously recorded ACK and NACK packets. Fortunately, a replayed ACK has no harmful effect apart from the futile energy usage due to the transmissions of the replayed ACK packet itself. This problem can be alleviated if the nodes keep track of the largest sequence number acknowledged so far, and drop ACK packets that refer to smaller sequence numbers, because in this case, a replayed ACK will be accepted only for a limited period (i.e., until the reception of a newer ACK or NACK containing a larger sequence number or base sequence number, respectively). A replayed NACK can also generate unnecessary re-transmission of data packets, but only until the eventual reception of a newer ACK or NACK packet that acknowledges the reception of those packets that still appear missing in the replayed NACK.

Considering the above discussion on possible attacks on control information and their effects, our main objective is to prevent the forgery and illegitimate alteration of control packets in DTSN. For this reason, our security extension proposed for DTSN hereafter aims at providing authentication and integrity protection for DTSN control packets.

Packet authentication services could be provided transparently for DTSN by lower layers, but there are at least two reasons why we cannot rely solely on security services provided by lower layers. First, as explained above, we want some level of protection against compromised intermediate nodes, and this requirement essentially excludes pure link level solutions. Although, we do assume message authentication services at the link layer in order to prevent the injection of fake data packets into the network, that service is not sufficient to detect forged control information injected by compromised intermediate nodes.

Second, the authentication and integrity protection of ACK and NACK packets must be provided in a way such that honest

intermediate nodes can also verify them, because in DTSN, intermediate nodes also process ACK and NACK information. This requirement calls for a broadcast authentication mechanism; however, broadcast authentication mechanisms are typically expensive either in terms of computational complexity (e.g., in case of asymmetric key digital signatures) or in terms of management overhead (e.g., in case of the symmetric key TESLA protocol [5]). Therefore, we do not want to provide a broadcast authentication service at the routing layer that is transparent to the upper layers, including DTSN, because this would mean to protect each and every upper layer protocol data unit, not only DTSN ACK and NACK packets, with the expensive broadcast authentication mechanism. Another reason why a transparent broadcast authentication service is not desirable for protecting DTSN control packets is that NACK packets are not only authenticated but they are sometimes also modified by intermediate nodes (set bits are cleared when the corresponding data packets are re-transmitted).

IV. The proposed authentication scheme

Our security extension to DTSN is based on symmetric key cryptographic primitives, and hence, it is efficient and easily applicable in the WSN context. In effect, our proposal is similar to the TESLA protocol [5] in spirit, but it does not require globally synchronized clocks in the nodes, neither it requires the establishment of any cryptographic context between the destination/source and the intermediate nodes on the route. The price that we pay for these advantageous features is that the proposed mechanism does not really provide a general purpose broadcast authentication service, but it is tailored for the specific problem of authenticating ACK and NACK packets. More specifically, our proposed mechanism ensures that an intermediate node can verify if an acknowledgment or negative acknowledgment information has really been issued by the destination, if and only if the intermediate node actually has in its cache the valid data packet referred to by the ACK or NACK. Note, however, that the real need to verify the authenticity of control information arises exactly in the case when the intermediate node has the referred data packet, because only in that case it needs to decide to delete it from the cache (in case of an ACK) or to re-transmit it (in case of a NACK). Otherwise, the intermediate node only needs to pass on the control information. Thus, in our scheme, forged control information can propagate in the network, but only until it hits an intermediate node that cached the corresponding valid data packet; this node can detect the forgery and drop the forged control packet.

The general idea of our protocol is the following: Each data packet is extended with two MAC (Message Authentication Code) values computed over the packet itself with two different keys, an ACK key and a NACK key, both specific to the data packet and known only to the source and the destination. When the destination wants to send an ACK referring to this data packet, it reveals its ACK key; similarly, when it wants to signal that this data packet is missing, it reveals its NACK key. Now, any intermediate node that has the data packet in question can verify if the ACK or NACK is authentic by checking if the appropriate MAC verifies correctly with the given key. As only the source and the destination can produce the right keys, but the source never reveals them, the intermediate node can be sure that the control information must have been sent by the destination. A side effect of the scheme is that the MAC values provide end-to-end protection, meaning that the destination can check the authenticity and integrity of each received data packet, which is also a desirable feature.

Below, we specify in details the management of ACK and NACK keys and the processing of ACK and NACK packets.

A. Key hierarchy

We assume that the source and the destination share a secret which we call the session master key, and we denote it by K. From this, they both derive an ACK master key K_{ACK} and a NACK master key K_{NACK} for the session as follows:

$$\begin{aligned} K_{ACK} &= PRF(K, \textit{“ACK master key”}, SessionID) \\ K_{NACK} &= PRF(K, \textit{“NACK master key”}, SessionID) \end{aligned}$$

where PRF is the pseudo-random function defined in [6] and $SessionID$ is the DTSN session identifier. The length of K, K_{ACK}, and K_{NACK} is 128 bits each.

The ACK key $K_{ACK}^{(n)}$ and NACK key $K_{NACK}^{(n)}$ for the n-th packet of the session (i.e., whose sequence number is n) are computed as follows:

$$\begin{aligned} K_{ACK}^{(n)} &= PRF(K_{ACK}, \textit{“per packet ACK key”}, n) \\ K_{NACK}^{(n)} &= PRF(K_{NACK}, \textit{“per packet ACK key”}, n) \end{aligned}$$

The length of the ACK keys is KL_{ACK} and the length of the NACK keys is KL_{NACK}. These are security parameters which will be analyzed later. Note that both the source and the destination can compute all these keys as they both possess the session master key K. Moreover, PRF is a one-way function, therefore, when the ACK and NACK keys are revealed, the master keys cannot be computed from them, and consequently, as yet unrevealed ACK and NACK keys remain secrets too.

In most applications, the destination of the data packets transferred with the help of DTSN is the base station. Under this assumption, the simplest way to generate the session master key between any DTSN enabled node and the base station is to derive it from a pre-established shared secret value, such as a node key shared by the node and the base station, which can be configured manually in the node before its deployment. Denoting the shared secret by S, the session master key K is then derived as follows:

$$K = PRF(S, \textit{“session master key”}, SessionID)$$

The advantage of this approach is that it does not even require an explicit message exchange between the source and destination for the purpose of establishing K.

In a more general setting, when a shared secret between the source and the destination cannot be assumed, we propose to use an authenticated Diffie-Hellman protocol variant such as

the Station-to-Station protocol [7]. In that case, an explicit message exchange is needed between the source and the destination which can precede the DTSN session establishment or it can be integrated with it.

B. Generating and processing ACK and NACK packets

As we said before, data packets are extended with two MAC values: these are called ACK MAC and NACK MAC. These values are computed from the entire DTSN packet with sequence number n using $K_{ACK}^{(n)}$ and $K_{NACK}^{(n)}$, respectively. The MAC function used for this purpose is HMAC [8]. The length of the ACK MAC is ML_{ACK} and the length of the NACK MAC is ML_{NACK}. These are security parameters which will be analyzed later. Both MAC values are placed in the DTSN header.

The ACK packet that refers to sequence number n is extended with the ACK key $K_{ACK}^{(n)}$. For this purpose the format of ACK packets is extended with an ACK key field. Similarly, the NACK packet with base sequence number n is extended with the ACK key $K_{ACK}^{(n)}$, as the semantics of the base sequence number in NACK packets is the same as that of the sequence number in ACK packets. In addition, if the i-th bit is set in the bitmap, then the NACK packet is further extended with the NACK key $K_{NACK}^{(n+i)}$. In order to accommodate these keys, the packet format of NACK packets is extended with an ACK key field and variable number of NACK key fields.

When an ACK packet is received by an intermediate node or the source, the node first checks if it has the corresponding data packet. If not, then the ACK packet is simply passed on to the next node towards the source. Otherwise, the node uses the ACK key obtained from the ACK packet to verify the ACK MAC value in the data packet. If this verification is successful, then the data packet can be deleted from the cache, and the ACK packet is passed on to the next node towards the source. If the verification of the MAC is not successful, then the ACK packet is silently dropped.

When a NACK packet is received by an intermediate node or the source, the node processes the acknowledgement part of the NACK packet as described above. In addition, it also checks if it has any of the data packets that correspond to the set bits in the bitmap of the NACK packet. If it does not have any of those data packets, it passes on the NACK without modification. Otherwise, for each data packet that it has and that is marked as missing in the NACK packet, it verifies the NACK MAC of the data packet with the corresponding NACK key obtained from the NACK packet. If this verification is successful, then the data packet is scheduled for re-transmission, the corresponding bit in the NACK packet is cleared, and the NACK key is removed from the NACK packet. After these modifications, the NACK packet is passed on to the next node towards the source.

In addition, each node maintains for each DTSN session the largest verifiably acknowledged sequence number so far, which we denote by $MaxSN$. If the sequence number n in a received ACK is smaller than or equal to $MaxSN$, then the ACK is silently dropped. If the base sequence number n in a received NACK packet is smaller than or equal to $MaxSN$, the NACK packet may still contain useful information in the bitmap. In particular, if the i-th bit is set and $n+i > MaxSN$ then the corresponding data packet needs to be re-sent (if the node has that data packet and verification of the corresponding NACK key is successful). However, if the i-th bit is set and $n+i \leq MaxSN$ then it can be cleared and the corresponding NACK key can be removed from the NACK packet. The full processing of NACK packets is illustrated in Figure 1.

V. Analysis

A. Security

The security of the proposed DTSN extension depends on the length of the MAC values ML_{ACK} and ML_{NACK}, and the length of the MAC keys, KL_{ACK} and KL_{NACK}. In general, the length of the MAC value determines the probability that an attacker can generate a valid MAC for a forged data packet by random guessing. If the length of the MAC is ML bits, then this probability is 2^{-ML}. The length of the key determines the resistance of the scheme to brute force key guessing attacks: the attacker can observe some data packets with valid MAC values, and try all possible keys until he finds one that produces the observed MAC values for all observed data packets. The average complexity of this attack, measured in the expected number of MAC computations, is 2^{-KL+1}, where KL is the key length in bits. Thus, larger MAC and key sizes are more secure, but on the other hand, MAC values add byte overhead to data packets, whereas ACK and NACK keys are part of control packets and therefore they add byte overhead to those packets. In particular, NACK packets may carry multiple NACK keys. Hence a good trade-off between security and overhead must be found.

In our case, producing a valid ACK MAC and NACK MAC on a forged data packet would result in the acceptance of the forged data packet by the destination, and the erasure of all potentially valid but not yet delivered data packets from the caches of the intermediate nodes up to the sequence number of the forged data packet. In order to prevent this attack, $ML_{ACK}+ML_{NACK}$ should be sufficiently large. Note, however, that the attacker cannot check off-line if a guessed MAC for a forged packet is correct, but it needs to send the forged packet to the destination, and see if it is accepted or not. In other words, the attacker must use the destination as an oracle. Moreover, he must prevent the destination from receiving the original data packet while he is submitting the forged packets with the same sequence number to the destination. This limits the power of the attack, and allows us to set the value of $ML_{ACK} + ML_{NACK}$ in the range of 40-64.

A brute force key guessing attack against our scheme would result in forged ACK and NACK information with all their consequences as discussed in Section III. In particular, an attacker can observe a data packet with its MAC values and try all possible keys until he finds the correct ACK and/or NACK key; then he can forge ACK and/or NACK packets. In order

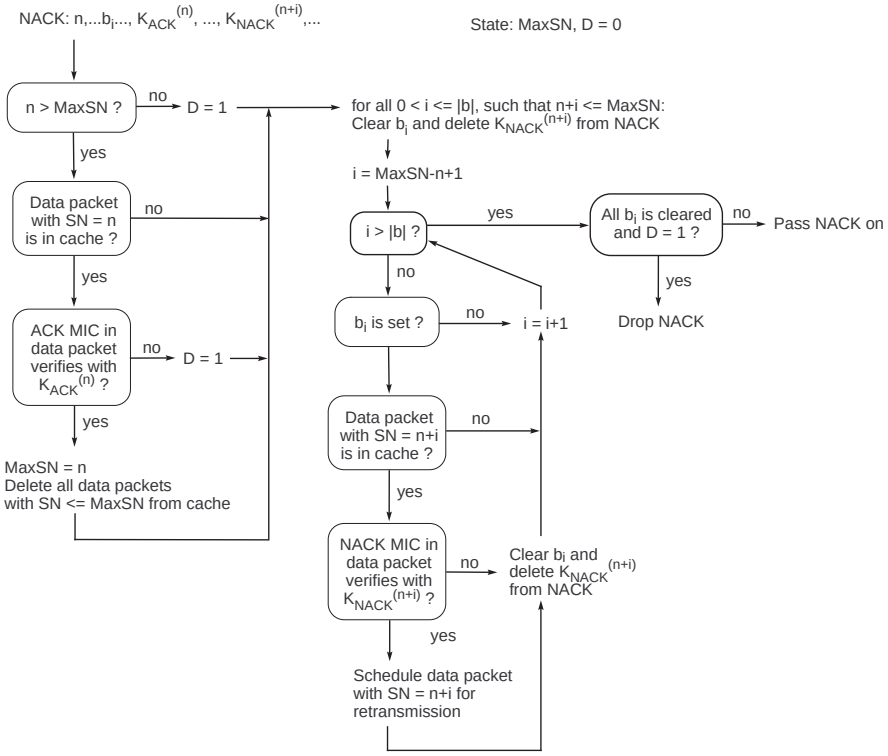

Fig. 1. NACK processing

to prevent this attack, both KL_{ACK} and KL_{NACK} should be sufficiently large. Note, however, that they do not necessarily need to be the same. Indeed, as forging ACK information has potentially more serious consequences, we may require that $KL_{ACK} > KL_{NACK}$.

There is also a subtle interplay between the MAC length and the key length in our scheme. Notably, it is not worth to choose the key length larger than the MAC length. The reason is that, in our scheme, the attacker actually does not need to find $K_{ACK}^{(n)}$ itself in order to forge an ACK on the n-th data packet, but any key with which the ACK MAC verifies correctly would be sufficient. In addition, it is sufficient if the guessed key works only for a single data packet. When the key length is KL and the MAC length is ML, and $KL > ML$, the expected number of keys with which the MAC in a given packet verifies correctly is 2^{KL-ML}. Hence, the complexity of a successful guessing attack against our scheme is $2^{KL}/2^{KL-ML} = 2^{ML}$.

Based on the above analysis, we propose the following parameter values: $ML_{ACK} = KL_{ACK} = 40$ and $ML_{NACK} = KL_{NACK} = 24$.

B. Overhead

The proposed authentication scheme imposes an additional protocol overhead, which decreases the energy-efficiency of the system. We evaluated the impact of the additional DTSN packet fields based on the analytical framework described in [9]. In this analytical framework, the DTSN end-to-end delivery procedure is represented as a Markov chain whose states correspond to the location of a packet on the path from the source to the destination at each iteration (i.e., at each (re-)transmission attempt). The end-to-end delivery cost is measured in terms of the expected total number of transmitted bytes (at the Media Access Control layer) required to accomplish the successful end-to-end delivery of a data packet. The model assumes that the Packet Error Rate (PER) is fixed and the same for all packets.

The presented numerical results assume that the used radio technology is IEEE 802.15.4. The considered PER values are within the realistic ranges reported in [10], with a maximum around 0.035 for indoor scenarios and 0.1 for outdoor scenarios in the absence of IEEE 802.11b/g interference, while with IEEE 802.11b/g interference, the worst-case PER may be as high as 0.80, even for very short communication distances (e.g., 5 m). These PER values already take into account the default number of retransmissions attempts, which is equal to 3.

Figure 2 shows the expected total number of transmitted bytes required to deliver a data packet with a payload of 60 bytes, as a function of the PER, with and without our proposed security extension.

As it can be seen, the overhead imposed by the security extension becomes more significant for PER values above 0.05. This can be explained by the fact that with the security extension, NACK packets become much longer since they must carry the NACK keys of the requested packets besides the sequence number bitmap. As the PER increases, the

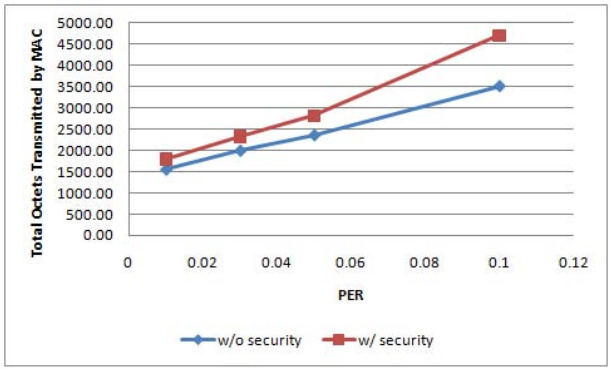

Fig. 2. Expected total number of transmitted bytes required to deliver a data packet with a payload of 60 bytes, as a function of the PER.

number of lost EAR and NACK packets (and consequently the number of EAR and NACK retransmissions) increases, which affects both the secured and the unsecured DTSN configurations. Additionally, the number of lost data packets also increases, and hence, the number of NACK keys in NACK packets, increasing the NACK length in the secured DTSN configuration. This explains the significant overhead difference observed in the results.

Figure 3 shows the expected total number of transmitted bytes required to deliver a data packet, as a function of the payload size, for the fixed PER of 0.1, with and without our security extension.

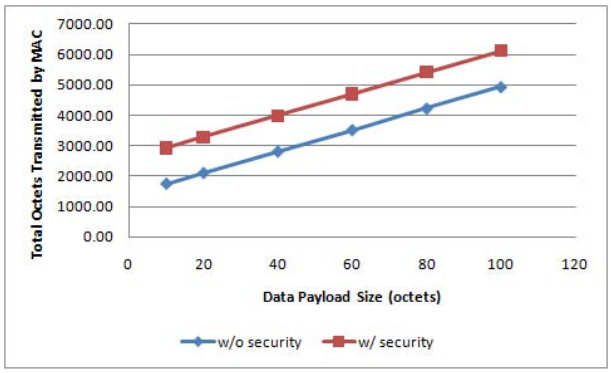

Fig. 3. Expected total number of transmitted bytes required to deliver a data packet, as a function of the payload size and PER = 0.1.

One can observe that the number of transmitted bytes increases linearly with the payload size. As expected, the security overhead is constant, which means that it is more significant for low payload sizes.

VI. Conclusion

In this paper, we proposed a security extension to the Distributed Transport for Sensor Networks (DTSN) protocol that transforms the otherwise insecure DTSN protocol into a secure WSN transport protocol that resists control packet manipulation attacks. Essentially, our security extension consists in a novel authentication scheme that prevents the forgery of transport control packets. As a side effect, our authentication scheme also provides end-to-end authentication and integrity protection for the data packets. The proposed security extension is solely based on symmetric key cryptographic primitives. We analyzed the overhead of the proposed scheme in terms of the amount of the additional bytes that need to be transmitted over the wireless medium, and our analysis showed that the overhead is bearable for realistic values of the Packet Error Rate (PER).

Our planned future work includes the implementation and validation of the proposed scheme in a WSN testbed environment. In addition, we intend to carry out a more detailed security analysis of the proposed protocol. So far, we assumed that compromised intermediate nodes attack only control packets, and the data packets cached by the honest intermediate nodes are intact. In a more general attacker model, however, compromised nodes may also modify or inject data packets. While those fake data packets would not be accepted by the destination, they do interfere with the mechanisms of our scheme. A detailed analysis and understanding of the effects of such attacks are also left for future work.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 225186. The first author has also been supported by the Hungarian Academy of Sciences through the Bolyai János Research Fellowship.

References

[1] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, pp. 921–960, 2007.

[2] C. Wang, K. Sohraby, B. Li, M. Daneshmand, and Y. Hu, "A survey of transport protocols for wireless sensor networks," *IEEE Network*, vol. 20, no. 3, pp. 34–40, 2006.

[3] L. Buttyan and L. Csik, "Security analysis of reliable transport layer protocols for wireless sensor networks," in *Proceedings of the IEEE Workshop on Sensor Networks and Systems for Pervasive Computing (PerSeNS)*, March 2010.

[4] B. Marchi, A. Grilo, and M. Nunes, "DTSN - Distributed Transport for Sensor Networks," in *Proceedings of the IEEE Symposium on Computers and Communications*, Aveiro, Portugal, 2007.

[5] A. Perrig, R. Canetti, D. Song, and D. Tygar, "The TESLA broadcast authentication protocol," *RSA Cryptobytes*, Summer 2002.

[6] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet RFC 5246, August 2008.

[7] A. Menezes, P. C. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[8] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Internet RFC 2104, February 1997.

[9] A. M. Grilo and N. M. Tiglao, "A Markov model of WSN reliable transport with cooperative caching," Technical Report, available on-line at http://www.inov.pt/WSN_Reliability_TR.pdf, June 2010.

[10] M. Petrova, J. Riihijärvi, P. Mähönen, and S. Labella, "Performance study of IEEE 802.15.4 using measurements and simulations," in *Proceedings of IEEE Wireless Communications and Networking Conference 2006 (WCNC'06)*, Las Vegas, USA, April 2006.