TCP Support for Sensor Networks

Torsten Braun[1], Thiemo Voigt[2], and Adam Dunkels[2]

Abstract— **Communication between sensors and controlling entities at the edge or outside the sensor network is needed for reliable remote sensor node management and reprogramming. TCP would be useful for tasks, where reliable unicast is appropriate, but the high bit error rates in wireless sensor networks lead to energy inefficiencies reducing the sensor network lifetime. We introduce an approach to support energy-efficient TCP operation in sensor networks. The concept called TCP Support for Sensor nodes (TSS) allows intermediate sensor nodes to cache TCP data segments and to perform local retransmissions in case of errors. TSS does not forward a cached segment until it knows that the previous segment has been successfully received by the next hop. This forms a backpressure mechanism for congestion control. Simulations show that TSS significantly reduces the number of TCP data segment and acknowledgement transmissions.**

Index Terms—**Transport Control Protocol, Wireless Sensor Networks, Congestion Control**

I. INTRODUCTION

Applications in wireless sensor networks (WSNs) typically require external connection to monitoring and controlling entities (sinks), which consume sensor data and interact with sensor devices. By running TCP/IP on the sensors a single standard protocol suite can be used and the sensor network can be directly connected to IP-based network infrastructures without proxies or middle-boxes. Data to and from the sensor network can be routed via any device with Internet connectivity (e.g. using GPRS) rather than via protocol proxy nodes only. Further, experiences from industrial researchers point out that using a standard such as TCP/IP for (on-body) WSNs has facilitated application development and system integration in terms of data collection and configuration [1]. While high packet loss rates and energy-inefficiencies limit the incentive of utilizing TCP in today's WSNs, we believe that advances in radio technology can decrease packet loss and energy harvesting can reduce the energy problems. Then, there will not be any reason not to use TCP/IP and abstain from a widely used standard that provides interoperability and is well understood by numerous practitioners and system administrators. In this paper we discuss problems with TCP/IP in today's WSNs and propose mechanisms that both turn TCP/IP into a viable option even for today's sensor networks and provide performance enhancements in tomorrow's WSNs.

While UDP may be used for transferring sensor data and other information that do not need reliable stream transport, TCP should be used for administrative tasks that require reliability and compatibility with existing application protocols. Examples for using TCP are configuration and monitoring of individual sensor nodes as well as download of binary code and data aggregation descriptions to sensor nodes. In particular, downloading code to designated nodes such as cluster heads in a certain geographical region requires a reliable unicast protocol. TCP can also be used for multicast communications based on overlay multicast using TCP for establishing overlay network links [22]. Tree topologies might have to be established then. This might also be applicable for convergecast scenarios such as for collecting sensor data.

In contrast to a common belief that TCP/IP implementations consume too many sensor resources, it has been shown that TCP/IP can be implemented on sensor nodes with limited processing power and memory [2]. TCP/IP may result in relatively large headers that may add significant overhead in case of short packets, but we assume that TCP is mainly used for configuration and programming tasks, where a rather high amount of data is transferred and packets become rather large. Nevertheless, we propose to develop a TCP/IP header compression scheme for sensor networks to address the header overhead issue. Due to the stateful approach proposed in this paper such a scheme should be feasible, but we leave this issue for future work. Other problems to be solved for TCP/IP in sensor networks are related to addressing. IP-based sensor networks may use spatial IP address assignment based on node locations, which might be relative to a base station location [3]. Data centric routing mechanisms are often preferable in wireless sensor networks [4]. To implement data centric routing in IP-based sensor networks, application overlay networks might be used.

TCP has been designed for wired networks with low bit error rates, interprets packet loss as congestion and thus decreases its transmission rate. Therefore, TCP has serious performance problems in wireless networks [5]. While this results in undesired but for this kind of networks not per se problematic low throughput, the main problem for sensor networks operating autonomously with constrained power supply is the energy-inefficient end-to-end retransmission scheme of TCP. In a multi-hop network, retransmitted packets must be forwarded by all intermediate nodes from sender to receiver, thus consuming energy at every hop. In general, end-to-end error recovery is not a good approach for reliable transport in sensor networks, because the per-hop packet loss rate may be in the range of 5% to 10% or even higher [6][9].

In this paper we introduce and evaluate TCP Support for Sensor nodes (TSS) [7], an approach that overcomes energy efficiency and performance problems. TSS is based on TCP data segment caching by intermediate nodes, local retransmissions after packet loss detection, aggressive TCP acknowledgement recovery, and backpressure congestion

1 University of Bern, Switzerland
2 Swedish Institute of Computer Science
Email: braun@iam.unibe.ch, thiemo@sics.se, adam@sics.se

control. TSS does not require any changes to TCP implementations at end points. While in [7] some of these mechanisms have been introduced and a TDMA-based network has been assumed, the contribution of this paper is the more detailed performance evaluation of the TSS mechanisms over a CSMA/CA based MAC layer,. The TSS concept has been improved and addresses problems that occur in CSMA/CA based WSNs. In particular, we investigate whether round trip time measurements between intermediate nodes and the destination can be used for setting retransmission timers. We also evaluate the usefulness of implicit acknowledgments that might require costly overhearing. Our simulations show that the overhearing time is short, making overhearing more energy-efficient than implementing explicit link level acknowledgements. A key contribution of this paper is the evaluation of the inherently included congestion control scheme, which automatically reduces the maximum congestion window to values proposed by related work. This is achieved without knowledge of the network topology.

While TSS focuses on TCP support, several protocols for reliable data transfer in sensor networks introduce new transport protocols and do not attempt to support TCP operation. Section II gives an overview about related work. We introduce TSS in Section III. Section IV describes performance results for reliable TCP data transfer across a multi-hop WSN using TSS. Section V concludes the paper.

II. Related Work

TSS extends ideas introduced by Distributed TCP Caching (DTC) [8], which is a generalization of the Snoop [5] protocol. DTC aims to avoid energy-costly end-to-end retransmissions by caching TCP data segments inside the network and retransmitting segments locally, i.e., from the intermediate sensor nodes' caches, when packet loss occurs. DTC assumes limited memory resources available for caching and proposes to cache a single segment per node. TSS mainly differs from DTC by the backpressure mechanism that keeps segments in the cache until a node knows that the previous segments have been received by the next hop node. This allows implementing a congestion control mechanism based on backpressure signals at the TCP sender. TSS does not use TCP options such as selective acknowledgements and retransmissions and hence requires less re-sequencing buffers at the receiver.

Reliable Multi-Segment Transport (RMST) [9] is used for sensor data transfer but not for control data transfer such as targeted by TSS. It can provide a caching mechanism within the intermediate nodes, but requires additional negative acknowledgement (NACK) messages. As a reaction on NACK messages, an upstream node can retransmit cached packets. RMST assumes a limited number of bytes in flight (< 5 KB) and that intermediate nodes can completely cache this amount of data. For packet loss rates below 10 % the combined caching and NACK mechanism is more efficient than pure link level ARQ approaches. On the other hand, processing of NACK messages in end points only is extremely inefficient for packet loss rates above 10 %. The design principles of TSS are consistent with these results. In contrast to RMST, TSS caching and local retransmissions rely on information from TCP packets only.

Pump Slowly Fetch Quickly (PSFQ) [6] is a reliable transport protocol for reprogramming sensor nodes. The pump operation aims to support quick forwarding in case of no errors and behaves like a store and forward approach for high error rates. It is based on broadcasting packets hop-by-hop from source to destination. While packet forwarding based on broadcasting has advantages in dynamic environments such as mobile ad-hoc networks and networks with unsynchronized sleep cycles [10], simulation experiments have shown that already a few packet losses due to congestion or bit errors can cause many duplicated packets resulting in unnecessary packet reception, processing and transmissions. The backpressure mechanism of TSS has a similar effect as the PSFQ pump operation: Packet forwarding will be slowed down as soon as errors are detected by the intermediate nodes. While PSFQ is focusing on code distribution using broadcast, TSS rather targets on communication with single nodes or smaller groups. PSFQ introduces NACK messages for proactively fetching retransmissions, while TSS supports standard mechanisms based on TCP acknowledgements and timeouts.

Event-to-Sink Reliable Transport (ESRT) supports reliable sensor data transport in WSNs [11]. It includes congestion control and mechanisms to achieve reliability based on status report transmissions from the sink back to the source. The frequency of the status reports depends on the observed and desired reliability as well as the needs from congestion control. As for PSFQ, a special protocol has been proposed, while no transport protocol extensions are required in TSS.

Congestion control is very important in WSNs, because overloading a WSN by too many transmissions can increase the collision probability. Collisions lead to packet loss as well as costly retransmissions. TCP congestion control limits the maximum window size according to slow start congestion control. However, it even might make sense to further limit the maximum congestion window, because the optimal window size in terms of throughput might be below the window size of standard TCP [12]. TCP's optimal throughput for a chain scenario is achieved when its window size is h/4 (h: number of hops). The backpressure mechanism used in TSS has a similar effect and limits the packets in transit, but without explicitly knowing the number of hops. Congestion Detection and Avoidance (CODA) [13] is based on congestion detection by monitoring channel utilization and buffer occupancy at the receiver. Detected congestion situations are signalled using backpressure signals towards the source, which then throttles down the transmission rate (open loop). Alternatively, sources regulate themselves on a longer time-scale based on acknowledgements received from the sink (closed loop). In contrast to TSS, CODA requires new signalling messages.

III. TCP Support for Sensor Nodes

TCP Support for Sensor nodes (TSS) aims to support energy-efficient operation of sensor nodes and forms a layer between TCP and the routing layer. TSS should ideally be implemented in TCP sensor nodes with senders and receivers as well as in intermediate sensor nodes that relay TCP data segments and acknowledgements of a TCP connection. TSS mechanisms do not require explicit link or MAC level acknowledgements, but only rely on TCP data segments and acknowledgements. This approach further reduces the amount of transmissions and can be used on top of any kind of sensor network MAC layer. By ensuring in sequence arrival of TCP data segments at the destination, TSS avoids any re-sequencing buffer and selective acknowledgement / retransmission extensions in TCP. TSS tries to reduce the number of transmissions by the following mechanisms.

A. Caching

An intermediate node caches a segment until it is sure that the successor node towards the destination has received the segment. A node knows this when it detects that the successor node has forwarded the segment (implicit acknowledgement) or when it spoofs a TCP acknowledgement that has been sent from the destination toward the source of the TCP data segment. Nodes are assumed to listen to packet transmissions of their neighbour nodes in order to be able to detect whether the neighbour nodes have forwarded TCP data segments. A packet that is known to be received by the successor node will be removed from the cache. In addition to the cache, TSS requires another (packet) buffer for temporarily storing the next packet that is waiting to be forwarded to the successor node. One might argue that forcing sensor nodes to overhear packets does not support energy efficient operation. An alternative would be explicit link level acknowledgements. However, this would not only require the node to listen and receive but also the successor node to transmit an additional acknowledgement packet. With our overhearing solution, a forwarding node should only listen to other's transmissions for a very short time. Typically, a packet will be forwarded immediately by the successor node and only in case of packet loss a node must overhear for the whole retransmission timeout interval. The performance evaluation in Section IV shows that the overhearing costs are very low.

B. Local Retransmissions of TCP Data Segments

All intermediate nodes are able to perform local retransmissions, when they assume that a cached segment has not been received by the successor node towards the destination. Retransmissions are triggered by carefully set timeouts. A retransmission timeout of 1.5 * rtt allows repairing even multiple packet losses before an end-to-end retransmission timeout is triggered. Simulations showed that a timeout of 2 * rtt performs slightly worse. The maximum number of local retransmissions has been limited to four. Bit errors or packet loss can cause duplicated packets in the network. Forwarding of duplicated packets can be prevented by a small history list consisting of the last few (here: ten) forwarded packets to filter out all segments that have been forwarded previously. TCP data segments can be uniquely identified by the source address and the IP identification field. End-to-end retransmissions should not be filtered in order to support end-to-end recovery in serious error situations.

C. Regeneration and Recovery of TCP Acknowledgements

TCP acknowledgements are very important for TSS, since several mechanisms such as round-trip-time estimation, retransmission, and caching depend on it. Experiments showed that loss of acknowledgements may have severe impact on the amount of TCP data segment transmissions. TSS deploys two mechanisms that help to decrease the number of TCP data segment transmissions significantly: a local TCP acknowledgement regeneration mechanism (like DTC) and an aggressive TCP acknowledgement recovery mechanism. The local TCP acknowledgement regeneration mechanism becomes active when a node receives a TCP data segment, which has already been acknowledged by the destination: The TCP data segment is dropped and a TCP acknowledgement with the highest acknowledgement number seen is regenerated and transmitted toward the source. The aggressive TCP acknowledgement recovery mechanism retransmits TCP acknowledgements, if a node has not discovered the forwarding of the TCP acknowledgements by the successor node. Since TCP acknowledgements should usually be forwarded without significant delay towards the sender of TCP data segments, each node measures the time between its own TCP acknowledgement transmission to the successor node and overhearing of TCP acknowledgement transmission from the successor node towards the TCP data segment sender (source). The TCP acknowledgement retransmission timeout is set to the double average value (using exponential averaging). After timeout expiration, a TCP acknowledgement is recovered using the highest acknowledgement number seen.

D. Backpressure Congestion Control

If the successor of a node has not forwarded all received packets, there might be a problem in the network. For example, the network might be congested or packet forwarding does not make progress, because a previously lost TCP data segment needs to be recovered first. If a node would continue with packet forwarding in case of congestion, the risk of unnecessary transmissions would be rather high. A forwarded segment might easily get lost then. The same is true in case of a lost packet due to bit errors. In such a situation all caches on subsequent nodes are occupied and the transmission of a new packet would not be protected by caching. For that reason, a TSS node stops any forwarding of subsequent packets until it knows that all earlier packets have been received and forwarded by its successor. Successful forwarding can be detected by overhearing the forwarded packet or by detecting a TCP acknowledgement for that TCP data segment. If packet forwarding stops at some point, all other nodes in the chain behind the stopping node will also stop their transmissions until progress is detected at their respective successor nodes. A lost packet (due to congestion or bit errors) should be recovered by the node that forwarded the packet at last. In that case, we have to avoid that retransmissions are triggered by nodes behind the recovering

node, i.e., the nodes closer to the sender. This can be achieved by increasing the retransmission timeouts at the nodes closer to the sender. The backpressure mechanism should be supported at the sender end point. We propose to not increase the TCP congestion window as long as there are a certain (here: three) number of packets waiting at the sender for transmission.

E. Pseudo Code

The operation of a TSS node in an intermediate system is presented in more detail by the C-like pseudo code below. The first part (1) describes acknowledgement timeout processing, i.e., when the node has not detected the forwarding of an acknowledgement by the next node towards the source. This implements the aggressive recovery scheme for TCP acknowledgements. The second part (2) shows processing of a TCP data segment retransmission timeout. Retransmissions are only performed, if the data to be retransmitted have not been confirmed by an implicit acknowledgement or by an explicit TCP acknowledgement. The main part (3) describes processing of received TCP acknowledgement and data segment packets. Part 3a describes normal processing, when a TCP data segment or a TCP acknowledgement has been received for forwarding. A newly received acknowledgement might confirm that some data have been received by the successor node. In that case, a segment waiting in the buffer might be forwarded by the node. The received acknowledgement might also stop an ongoing rtt measurement. If the acknowledgement acknowledges previously acknowledged data again, we drop it, but forward it towards the source otherwise. Data processing in part 3a is applied to packets that need to be forwarded towards the destination. If there is a gap between the packet's sequence number and the sequence number of the highest byte transmitted, the packet is discarded. Otherwise, if there is a gap between the packet's sequence number and the sequence number that the successor node has received, the packet needs to be stored in the buffer before it can be forwarded. The packet may also include data that has all been acknowledged by the destination. In that case, it is not forwarded further, but an acknowledgement is regenerated and sent towards the source. If all transmitted data have been confirmed and the packet contains the next unconfirmed byte, the packet can be forwarded immediately and a new rtt measurement might be started if such a measurement is not yet going on. Part 3b shows processing of an overheard packet. In case of an acknowledgement, the acknowledgement timer is cancelled and the time needed by the upstream node to forward an acknowledgement is measured in order to calculate the acknowledgement retransmission timeout. For an overheard data packet that has been cached, the retransmission timer is cancelled as well and the cache is released. If there is another packet waiting in the buffer, it will be forwarded if it is eligible. However, the forwarding must be delayed randomly in order to reduce the risk of collisions. Simulations have shown that immediate forwarding significantly increases collision probability.

```
switch(event){
  case ack_timeout: // -1-
    retransmit_ack(acknowledged);
    start(ack_timer, acknowledged,
      γ^attempts++ * ack_forwarding_time);
    break;
  case retransmission_timeout: // -2-
   sequence_no =
    sequence_number_of_packet_to_be_retransmitted;
    if ((sequence_no + length > confirmed){
     retransmit_data(sequence_no);
     if (number_of_retransmissions > limit)
        delete(cache);}
    break;
  default: // -3-
   if (packet_has_bit_error || ttl_expired ||
   (own_address != next_address) &&
   (own_address != previous_address))
     delete(packet);
   else if (next_address == own_address) { // -3a-
     switch (type_of_packet){
      case ack:
       acknowledged=max(ack_no- 1, acknowledged);
       if ((acknowledged > confirmed) &&
        ((byte[acknowledged+1]∩buffered_packet)
            ≠ Ø)){
         forward(buffered_packet);
         move(buffered_packet, cache);
         transmitted=
           sequence_number_of_buffered_packet +
           length-1;
         start_timer(retransmission_timer,
            sequence_no, β * rtt);
         confirmed = acknowledged;}
       if (ongoing_rtt_measurement &&
        (ack_no > rtt_sequence_no)){
         rtt= (1-α) * rtt + α *
           (current_time-start_of_measurement);
         ongoing_rtt_measurement = FALSE;}
       if (ack_no <= ack_forwarded)
         delete(packet);
       else {
        forward(packet);
        start(ack_timer, ackno,
           γ * ack_forwarding_time);
        attempts = 1;}
       break;
      case data:
       if (sequence_no > transmitted + 1)
        delete(packet);
       else if ((sequence_no > confirmed + 1) &&
        (buffer_is_empty ||
        (sequence_no < seqno_of_buffer)))
         move(packet, buffer);
       else if (sequence_no + length – 1
         <= acknowledged){
        retransmit_ack(acknowledged);
        start_timer(ack_timer,acknowledged,
         γ*ack_forwarding_time);
        attempts = 1;
        delete(packet);}
       else if ((transmitted == confirmed) &&
        (byte[confirmed + 1] ∩ packet) ≠ Ø)){
         if (! ongoing_rtt_measurement){
          ongoing_rtt_measurement = TRUE;
          rtt_sequence_no = sequence_no;
          start_of_measurement = current_time;}
          forward(packet);
          transmitted = sequence_no + length - 1;
          move(packet, cache);
          start_timer(retransmission_timer,
            sequence_no, β * rtt);}
         else
          delete(packet);}
```

```
   else if (own_address == previous_address){ //-3b-
    switch (type_of_packet){
     case ack:
      ack_forwarding_time = (1 - α) *
        ack_forwarding_time + α *
       (current_time - transmission_time(ack_no));
        cancel(ack_timer, ack_no);
        ack_forwarded = ackno;
        break;
     case data:
      if (sequence_no + length - 1 > confirmed){
          cancel(retransmission_timer,
sequence_no);
       delete(cache);
       confirmed = sequence_no + length - 1;
       if (byte[confirmed + 1]∩buffered_packet≠∅){
        forward_delayed(buffer);
        transmitted = sequence_no_of_buffered_packet
         + length - 1;
        move(buffer, cache);
        start(retransmission_timer,
          sequence_no_of_buffer, β * rtt);}}}
  delete(packet);}}
```

IV. Performance Evaluation

We have evaluated TSS using the Omnet++ simulator [14]. The simulation scenario is a chain of eleven nodes with a distance of 200 m between each node. A transmission range of 200 m is feasible in outdoor environments with various sensor nodes such as ESB [15] or WiseNet [16] nodes. The chain scenario is a rather typical scenario in sensor networks, when a sink needs to configure a single node. Cross traffic does not occur, if there is a single sink communicating with a single node or a group of nodes at one instant. For multiple TCP connections in a multicast overlay network we expect interference rather at the sink or branch nodes. Moreover, TCP connections may compete with sensor data flows from sources to sinks. Interference issues are left for future work. The TCP sender implementation (node 10) and the TCP receiver implementation (node 0) exchange 1000 TCP data segments with a payload size of 1000 bits plus TCP/IP and MAC header (= 20 + 20 + 12 bytes = 416 bits). For throughput and packet transmission measurements we performed 100 simulation runs per experiment with 1000 TCP data segment transmissions from source to destination. For local rtt measurements, overhearing time evaluation, and congestion control considerations we performed a single simulation run with medium bit error rate.

While the results in [7] are based on simulations featuring a simple TDMA MAC layer, our TSS implementation running on each node includes a CSMA/CA MAC implementation, which senses the transmission medium and backs off in case of a busy medium. In order to save energy we back off without sensing the medium for a random time between 1τ and 3τ with τ = time to transmit a 1000 bit payload TCP data segment. Furthermore, we assume equal transmission power of all senders. A receiver can correctly receive a packet from a sender if it is not further away than 200 m and the signal to noise ratio is less than 10 dB. A receiver can detect an ongoing transmission if it receives a signal that is equivalent to a sender 500 m away. Intentionally, we did not implement an RTS/CTS collision avoidance scheme, since such a scheme may be very costly, create 40 % overhead and may not avoid all collision situations [17]. Since the RTS/CTS mechanism doubles the number of packet / acknowledgement transmissions, we propose to avoid collisions on a higher layer than MAC level. For example, if a node has recently forwarded a segment to the receiver, subsequent segments should not be forwarded immediately but slightly delayed. We implemented such a collision avoidance scheme in TSS. This approach is somewhat similar to the adaptive rate control scheme proposed in [17]. In our simulations we did not put the nodes into any sleep mode. Integration with sleep / awake scheduling is also left for future work, although coordinated sleeping as proposed by S-MAC [21] could easily be integrated. Coordinated sleeping would not affect our proposed mechanisms. We also assume that the MAC layer does not use explicit acknowledgements, because they are considered as too costly. The bit rate of the wireless network is 100 kbps. Moreover, we assume that a node considers an overheard TCP data segment as correctly received, if the TCP/IP and MAC header (416 bits) has been received without error. We investigated certain uniformly distributed bit error rates [18], in particular no (0), low (10^{-6}), medium (10^{-5}) and high (10^{-4}) bit error rates. Such uniformly distributed error models are rather disadvantageous for our scheme, since a single bit error temporarily stops packet forwarding in a chain of nodes. The bit error rates used result in up to 15 % packet errors. Similar packet error rates have been used in [9] and measured for connected networks in [19].

A. Packet Transmissions

The number of packet transmissions is the most important metric, because the energy efficiency strongly depends on it. Table 1 shows the number of TCP data segment and acknowledgement transmissions for different bit error rates. For TSS we used two variants: In the first variant (backpressure in end point) we combined the backpressure mechanism with the TCP congestion control. The congestion window is increased after receiving a TCP acknowledgement, if there are more than a certain number (here: three) TCP data segments waiting for transmission at the source node. In the second variant (maximum congestion window = 3), we limited the maximum congestion window dependent on the number of hops according to [12]. The first TSS variant resulted in better throughput performance, but required slightly more packet transmissions. Note that the first variant is independent from the topology, but has a similar effect as the congestion window limit used in the second variant. In particular for higher bit error rates the throughput improvement is higher. For high bit error rates several packets might be cached and buffered in the sensor network, but might wait for forwarding due to the timeout based retransmission mechanisms. Therefore, we consider the first variant as a better choice.

Due to the CSMA/CA MAC layer, there are always a certain but low number of collisions that result in corrupted packets. Therefore, TSS already performs better than TCP for zero and low bit error rates. In particular for medium and high bit error rates, the difference in packet transmissions between

TSS and TCP becomes evident (cf. Fig. 1). The main reasons for the high number of packet transmissions required for TCP without running TSS in the sensor nodes are the many end to end transmissions. The optimal number of TCP data segment transmissions is calculated by

$$\frac{1}{1-PER} \bullet (number\ of\ packets\ for\ PER = 0),$$

$PER = 1-(1-BER)^n$, BER / PER: bit / packet error rate.

The difference between the TCP data segment transmissions using TSS and the optimal number is low for all bit error rates. Considering that retransmissions due to collisions are not counted for the optimal number of transmitted TCP data segments, TSS performs nearly optimal for all bit error rates. DTC and TSS performance have been compared in [7] using a collision free TDMA MAC layer. For packet error rates below 5 % DTC and TSS have a similar number of total packet transmissions, while the total number of packet transmissions is somewhat lower for TSS in case of packet error rates above 10 %. TSS results in a lower number of TCP data segments, while the number of acknowledgements is always higher compared to DTC, because of the aggressive acknowledgement recovery scheme implemented in TSS.

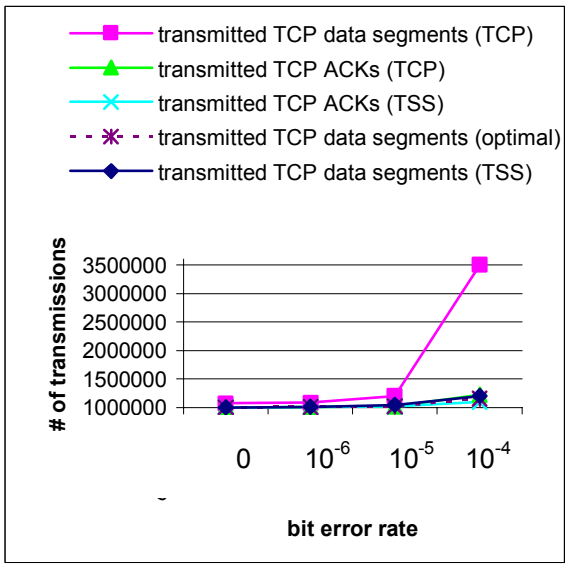

Fig. 1: Packet transmissions of TCP and TSS

TABLE 1. PACKET TRANSMISSIONS AND THROUGHPUT

Bit error rate	**0**	**10^{-6}**	**10^{-5}**	**10^{-4}**
TCP				
transmitted TCP data segments	1067600	1081090	1197001	3499974
transmitted TCP ACKs	1001000	1003015	1019395	1217739
total number of packets	2068600	2084105	2216396	4717713
e2e retransmissions	33300	34494	45991	474776
throughput [bps]	1955	1811	831	7
TSS (backpressure in end point)				
transmitted TCP data segments	1002061	1016829	1058486	1231501
transmitted TCP ACKs	1001600	1000467	1002887	1075384
total number of packets	2003661	2017296	2061373	2306885
e2e retransmissions	0	146	233	1552
throughput [bps]	4997	4412	2969	465
TSS (maximum congestion window = 3)				
transmitted TCP data segments	1002061	1011693	1046849	1200717
transmitted TCP ACKs	1001600	1003210	1015297	1092203
total number of packets	2003661	2014903	2062146	2292920
e2e retransmissions	0	199	417	1909
throughput [bps]	4997	4309	2626	288
Optimal number of transmitted TCP data segments	1001500	1002919	1015782	1153852

B. Throughput

Although we did not optimize TSS for throughput - the main goal was to keep the number of transmissions as low as possible - TSS throughput is always significantly higher than for TCP. For zero or low bit error rates we achieve a throughput of nearly 5 kbps. Compared with the network bandwidth of 100 kbps, this is a reduction by a factor of 20. First, we have to take into account that TCP acknowledgements consume a high fraction of the capacity and the TCP/IP/MAC header overhead is high. Each payload byte causes nearly another byte to be transmitted in the header part of the TCP data segment or the TCP acknowledgement. This could be improved by TCP/IP header compression. Second, packets need to be forwarded 10 times and spatial reuse is rather limited in our investigated scenario. Typically, two nodes can send simultaneously. Therefore, we can not expect a total throughput of more than 10 kbps. A further throughput decrease is caused by the delay of the CSMA/CA MAC scheme, collisions, and TCP congestion control. TSS has rather low throughput decrease up to the medium bit error rate, while plain TCP drops significantly already for medium bit error rates.

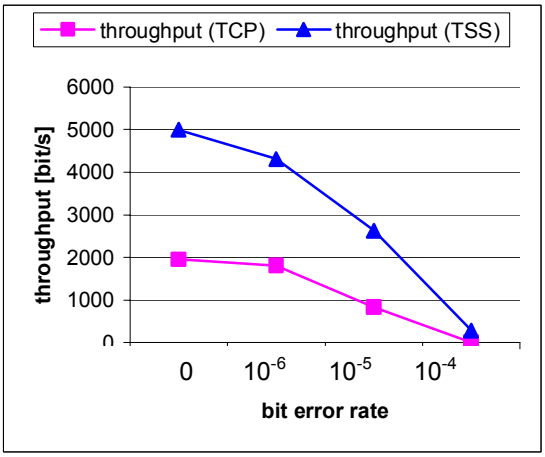

Fig. 2: Throughput of TCP and TSS

For high bit error rates the packet error rate is approximately 14 % per link. In that case, nearly every packet is dropped for TCP without TSS on the path from source to destination. TCP throughput is therefore close to 0, while TSS can at least achieve some low throughput (cf. Fig. 2). For such high bit error rates, packet sizes could be decreased in order to decrease the packet error rate for a given bit error rate.

C. Local RTT Measurements

The local retransmission scheme deployed at TSS nodes is similar to DTC and depends on the estimation of the round trip time (rtt) between the node and the destination. The retransmission timeout is set to 1.5 * rtt, while the rtt is calculated using exponential averaging of rtt samples. To support fast convergence, we initialize the rtt value by the delay measured during a SYN/SYNACK exchange during TCP connection establishment. Fig. 3 shows that the average rtt values used for retransmission timeout calculation decrease at the nodes that are closer to the destination and further away from the source. This is exactly the behaviour we need for both the local retransmission and backpressure schemes.

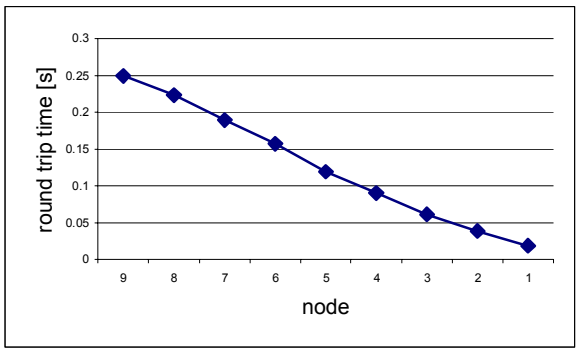

Fig. 3: Average rtt at nodes 9-1 (node 1 = neighbor of destination)

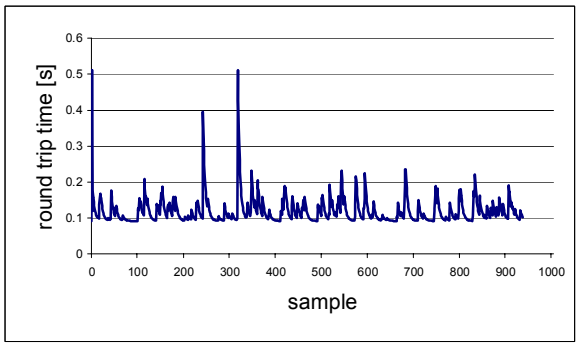

Fig. 4: rtt samples at node 5

Fig. 4 shows rtt development at node 5 (in the middle between source and destination) for a single simulation run (exchange of 1000 packets) and medium bit error rate. All simulations for rtt measurements have been performed using TSS with backpressure in the end point. The same variant has been used in the following subsections too.

D. Overhearing Time

Another issue to be investigated further is the problem that has been caused by using implicit acknowledgements. After a node has forwarded a packet it needs to overhear its successor's transmission. This requires a node to stay in idle state and prevents it from going into sleep state. In the worst case, a node needs to listen for the time interval for which a packet is stored in the cache. This time is limited by the retransmission timeout interval. We measure the time a packet is stored in the retransmission buffer until the transmitted packet is either overheard or the retransmission timeout expires. This time includes at least two packet transmissions, i.e., the transmission from the first node to its successor and the transmission by the successor node. For a packet size of 1416 bits and 100 kbps link, this time must be at least 28 ms plus a small back-off time. Fig. 5 plots the cumulative distribution function for these times. To get the results we performed again a single simulation run transmitting 1000 packets with medium bit error rate and measured the time values at node 5. We see that in 97 % of the cases, the packet is overheard after approximately 28 ms. However, due to packet loss and retransmission timeout expirations, the time values go up to 280 ms, but in average a node must store the packet 33.5 ms only, which is less than 20 % above the minimum value. Link level acknowledgements may be an alternative to overhearing. However, if we assume that transmitting a bit is 50 % more costly than receiving or overhearing a bit, transmission of link level acknowledgements with 24 bytes (= 1416 bits * 20 % / 1.5 / 8) costs more energy than overhearing.

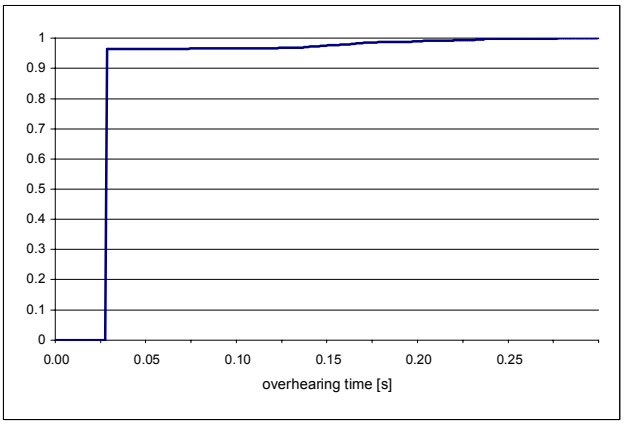

Fig. 5: Overhearing time (cumulative distribution function)

E. Packets in Flight

The backpressure based congestion control limits the maximum congestion window for TSS to approximately 15 in all investigated scenarios. However, the number of packets in flight is much lower. We measure the number of packets in flight after each segment has been sent by determining how many segments did not yet arrive at the receiver. Fig. 6 and Fig. 7 show histograms for the number of packets in flight between the sender and the receiver for a scenario with 11 and 21 hops respectively. We see that there are less packets in flight for a lower number of hops. The values for the packets in flight are in most cases lower than h/4 (h = number of hops), which is the optimal value for the TCP window size in a multi-hop chain scenario [12]. The average values are 1.4 and 2.5 respectively. This shows that the backpressure mechanism effectively limits the number of packets in flight to a similar number that has been proposed by other related work on congestion control in multi-hop wireless networks. Note that in our case, we do not have to know the number of hops between sender and receiver, but the backpressure mechanism adapts automatically to an appropriate value!

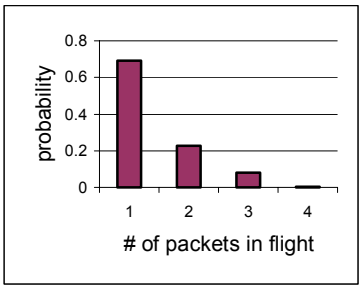

Fig. 6: Number of packets in flight for 11 hops

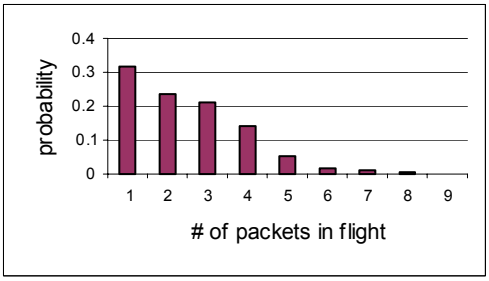

Fig. 7: Number of packets in flight for 21 hops

F. *Memory Consumption*

Another issue is the memory consumption in a sensor node to support TSS. For each supported TCP connection, we need memory space for 2 packets. The history list for detecting packet duplicates requires 2 D 32 bit variables, if we store the last D packets. Moreover, we need 12 further 32 bit variables for storing state information such as sequence numbers. For D = 10, this adds up to 128 bytes. In total a TCP connection needs 128 bytes plus the memory for storing two TCP data segments. The memory limitations of a node will limit the number of concurrent TCP connections to be supported.

V. CONCLUSIONS

TCP support in WSNs is desirable to allow direct communication of sensor nodes with other systems for various purposes such as configuration, reprogramming or management. This paper showed that even in scenarios with high error rates, TCP can be used in an energy-efficient way, if some protocol support is provided in intermediate nodes. The proposed mechanisms drastically reduce the number of TCP data segment transmissions needed to transfer a certain amount of data across a WSN with relatively high bit / packet error rates. Moreover, we have evaluated a novel congestion control mechanism that is very effective as well as easy to implement and deploy. Future work will analyse the performance in more complex network scenarios such as tree structures and consider background data traffic from sensors to the sink as well as multiple TCP connections. Additional work needs to be done for considering more complex bit error patterns [20]. We also plan to integrate scheduling mechanisms for sleep cycles and consider real implementation on real sensor nodes. Further reduction of transmissions might be achieved by combining data and acknowledgement transmissions. Also, packet sizes need to be adapted for very high bit error rates. We also propose header compression for reducing the header overhead of TCP data segments and acknowledgements.

VI. REFERENCES

[1] A. Chrisitan, J. Healey: Gathering Motion Data Using Featherweight Sensors and TCP/IP over 802.15.4, Workshop on On-Body Sensing, Osaka, Japan, October 2005.

[2] A. Dunkels: Full TCP/IP for 8-bit Architectures, ACM MobiSys, pp. 85-98, San Francisco, May 2003.

[3] A. Dunkels, T. Voigt, J. Alonso: Making TCP/IP Viable for Wireless Sensor Networks, Work in Progress Session at 1st European Workshop on Wireless Sensor Networks (EWSN 2004), Berlin, January 2004

[4] D. Estrin, R. Govidan, J. Heidemann S. Kumar. Next century challenges: scalable coordination in sensor networks, Mobile Computing and Networking, pp. 263-270, 1999.

[5] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz. Improving TCP/IP performance over wireless networks. ACM Mobicom, pp. 2-11, November 1995.

[6] C.-Y. Wan, A. Campbell, L. Krishnamurthy: PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks, 1st ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, September 28, 2002

[7] T. Braun, Th. Voigt, A. Dunkels: Energy-Efficient TCP Operation in Wireless Sensor Networks, Praxis der Informationsverarbeitung und Kommunikation (PIK), No. 2, 2005.

[8] A. Dunkels, T. Voigt, H. Ritter, J. Alonso: Distributed TCP Caching for Wireless Sensor Networks. Annual Mediterranean Ad Hoc Networking Workshop, Bodrum, Turkey, June 2004.

[9] F. Stamn, J. Heidemann: RMST: Reliable Data Transport in Sensor Networks, 1st IEEE International Workshop on Sensor Net Protocols and Applications, Anchorage, May 11, 2003

[10] M. Heissenbuüttel, T. Braun, M. Wälchli, Th. Bernoulli: Optimized Stateless Broadcasting in Wireless Multi-hop Networks, IEEE Infocom 2006, Barcelona, April 25-27, 2006

[11] Y. Sankarasubramanian, Ö. Ankan, I. F. Akyildiz: ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks, ACM MobiHoc, Anaheim, June 1-3, 2003

[12] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla: The Impact of Multihop Wireless Channel on TCP Throughput and Loss, IEEE Infocom, San Francisco, March 30 - April 3, 2003

[13] C.-Y. Wan, S. Eisenman, A. Campbell: CODA: Congestion Detection and Avoidance in Sensor Networks, ACM SenSys, Los Angeles, November 3-5, 2003

[14] Omnet++: Discrete Event Simulation System, web page, visited 2004-11-21, http://www.omnetpp.org

[15] J. Schiller, A. Liers, H. Ritter, R. Winter, T. Voigt: ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing, Hawaii International Conference On System Sciences (HICSS 2005), Hawaii, January 2005

[16] C. Enz, A. El-Hoiydi, J.-D. Decotignie, V. Peiris: WiseNET: An Ultra-Low Power Wireless Sensor Network Solution, IEEE Computer, August 2004, pp. 62

[17] A. Woo, D. Culler: A Transmission Control Scheme for Media Access in Sensor Networks, ACM Mobicom, Rome, 2001

[18] A. Gurtov, Sally Floyd: Modeling Wireless Links for Transport Protocols, ACM SIGCOMM CCR, Vol. 34, No. 2, April 2004

[19] A. Woo, T. Tong, D. Culler: Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks, ACM SenSys, 2003, November 5-7, 2003, Los Angeles

[20] A. Köpke, A. Willig, H. Karl: Chaotic Maps as Parsimonious Bit Error Models of Wireless Channels, IEEE INFOCOM, San Francisco, California,USA, March 2003.

[21] W. Ye, J. Heideman, D. Estrin: Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks, IEEE/ACM Transactions on Networking, June 2004, pp. 495

[22] F. Baccelli, A.Chaintreau, Z. Liu, A. Riabov: The one-to-many TCP overlay: a scalable and reliable multicast architecture: IEEE Infocom 2005, March 13-17, 2005