Energy-Efficient TCP Operation in Wireless Sensor Networks

Torsten Braun
Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland
braun@iam.unibe.ch

Thiemo Voigt, Adam Dunkels
Swedish Institute of Computer Science, Kista, Sweden
{thiemo, adam}@sics.se

Abstract

Many applications of wireless sensor networks require connectivity to external networks to let monitoring and controlling entities communicate with the sensors. By using the TCP/IP protocols inside the sensor network, external connectivity can be achieved anywhere in the sensor network. In such IP-based sensor networks, TCP can be used for remote management and reprogramming of sensor nodes. However, the high bit error rates in multi-hop sensor networks lead to energy-inefficiencies that reduce the lifetime of the sensor network. This paper introduces and compares two approaches to support energy-efficient operation of TCP in sensor networks: Distributed TCP Caching (DTC) and TCP Support for Sensor networks (TSS). Both concepts allow intermediate sensor nodes to cache TCP segments and to perform local retransmissions in case of errors. This allows reducing the total number of packet transmissions in the sensor network when transferring data to or from a sensor node. DTC caches and immediately forwards TCP data segments, whereas TSS does not forward a cached segment until it knows that the previous segment has been successfully received by the next hop node. We show by simulation that both approaches significantly reduce the number of TCP segment and acknowledgement transmissions. Their performance differs slightly depending on the error rate. Both approaches have also slightly different needs in buffer requirements and TCP options to be supported.

Keywords

Wireless sensor networks, transport layer, reliable data transport, TCP

1. Introduction

Wireless sensor networks are composed of a large number of radio-equipped sensor devices that autonomously form networks through which sensor data is transported. The devices are typically severely resource-constrained in terms of energy, processing power, memory, and communication bandwidth. Many applications of wireless sensor networks require an external connection to monitoring and controlling entities that consume sensor data and interact with the sensor devices. Running TCP/IP in the sensor network makes it possible to connect the sensor network directly to IP-based network infrastructures without proxies or middle-boxes. Since each sensor device is able to communicate using TCP/IP, it is possible to route data to and from the sensor network using standard IP-based technologies such as General Packet Radio Service (GPRS).

Data transport in IP-based sensor networks is performed using the two main transport protocols in the TCP/IP stack: the best-effort UDP and the reliable byte-stream protocol TCP. UDP is used for sensor data and other information that do not use unicast reliable byte-stream transmission. TCP should be used for administrative tasks that require reliability and compatibility with existing application protocols. Examples of such tasks are configuration and monitoring of individual sensor nodes, downloads 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.

It is well known that TCP has serious performance problems in wireless networks [8]. One problem is that TCP, which has been designed for wired networks with low bit error rates, interprets packet loss as an indication of congestion and decreases its transmission rate in case of a lost packet. This leads to low throughput, which is a major problem in mobile ad-hoc networks that are typically operated by human users. For sensor networks, which operate autonomously with constrained power supply, the main problem is the energy-inefficiency of TCP. This energy-inefficiency is caused by TCP's end-to-end retransmission scheme, which requires that lost packets are retransmitted by the original sender of the packet. In a multi-hop network, the retransmitted packet must be forwarded by all intermediate nodes from the sender to the receiver, thus consuming valuable energy at every hop. In general, end-to-end recovery is not a good candidate 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 [10].

In this paper we compare two different approaches that overcome these problems: Distributed TCP Caching (DTC) [1] and TCP Support for Sensor networks (TSS). Both schemes work by letting intermediate nodes cache TCP data segments and perform local retransmissions when packet loss is detected. Neither DTC nor TSS requires any changes to TCP or the TCP implementations at the end-points. In TSS, a caching node will not forward a cached TCP segment until it knows that the next-hop node has received all TCP segment with lower sequence numbers. This causes a backpressure that is intended to reduce the number of packet transmissions. DTC, on the other hand, immediately forwards all TCP segments, even those that are cached in the intermediate nodes. In addition, TSS uses an aggressive TCP acknowledgement recovery mechanism.

Our results show that both DTC and TSS significantly enhance TCP performance both in terms of the overall number of transmitted TCP segments and the number of end-to-end retransmissions. Due to the aggressive TCP acknowledgment recovery scheme in TSS the total number of TCP acknowledgements is higher for TSS than for DTC. On the other hand, the number of transmitted TCP data segments is lower for TSS because of the backpressure mechanism. For low packet loss rates, DTC and TSS have a similar total number of packet transmissions, whereas DTC incurs more packet transmissions than TSS for high packet loss rates.

In addition to the energy-efficiency issues of TCP, there are several problems that need to be solved before TCP/IP can be used efficiently in wireless sensor networks. In Section 2 we describe these problems and outline potential solutions. Several protocols for reliable data transfer in sensor networks have been proposed. However, most approaches introduce new transport protocols and do not attempt to support TCP operation. Section 3 gives an overview about related work in this area. We propose to use TCP for reliable data transfer in sensor networks and propose the DTC and TSS mechanisms in Sections 4 and 5. Section 6 compares performance results for transferring a certain amount of data across a multi-hop wireless sensor network. Section 7 concludes the paper.

2. IP-based Sensor Networks

Besides poor TCP performance both in terms of throughput and energy-efficiency, there are other problems with TCP/IP that must be solved before IP-based sensor networks can become ubiquitous. In this section we identify these problems and sketch solutions. We discuss these in more detail in [2].

Node limitations. In order for wireless sensor networks to be feasible, each sensor node is typically limited in terms of memory and processing power. It has often been assumed that a TCP/IP stack is too heavy-weight for such a small system. However, previous work [4] shows that this is not the case. Our uIP implementation of the TCP/IP stack [5] runs on 8-bit micro-controllers requiring only a few hundred bytes of RAM.

Address centric addressing. The IP addresses in traditional IP networks are assigned to each network interface based on the network topology. Each network interface is assigned a unique IP address using either manual configuration or semi-automated mechanisms such as DHCP. Such address assignment mechanisms are not suited for large scale sensor networks. Instead, IP-based sensor networks may perform *spatial IP address assignment* that uses the spatial location of the sensor nodes to construct semi-unique IP addresses.

Address centric routing. In traditional IP networks, each packet is transparently routed through the network. The routing path is based on the IP addresses and the topology of the network. For wireless sensor networks, data centric routing mechanisms are often preferable [7]. To implement data centric routing in IP-based sensor networks, we use *application overlay networks*.

Header overhead. Compared to specialized sensor networking protocols, the protocols in the TCP/IP suite have a very large header overhead. The shared context nature of sensor networks enables efficient *header compression* to reduce TCP/IP header overhead.

We have implemented and demonstrated an IP-based sensor network [3] consisting of Embedded Sensor Boards (ESBs) [6] running the uIP stack. This network used spatial IP addressing and performed application overlay routing.

3. Related Work

DTC and TSS are inspired by the Snoop [8] protocol that has been developed for supporting TCP over wireless access networks. The Snoop agent is deployed at an intermediate system between the wireless and wired part of the network. The agent buffers TCP segments that have not yet been acknowledged by the receiver and detects TCP segment loss by analysing TCP acknowledgements. In that case, the agent can perform local retransmissions as well as suppress TCP acknowledgements in order to avoid duplicate acknowledgments at the sender. Duplicate acknowledgements might cause end to end retransmissions for packets that could also be recovered locally by the agent. DTC can be considered as a generalization of Snoop for wireless multi-hop networks, while TSS further reduces the number of transmissions and adds mechanisms to throttle the transmission rate in case of packet losses.

Reliable Data Transport in Sensor Networks (RMST) [9] has been designed for the use together with directed diffusion. RMST is used for sensor data transfer but not for control data transfer as our mechanisms. It can provide a caching mechanism within the intermediate nodes, but requires additional negative acknowledgement (NACK) messages. These are sent by an intermediate node to its upstream neighbour, when it detects, e.g. using timeouts, holes in the data flow. As a reaction on NACK messages, an upstream node can retransmit cached packets. The authors assume a low number of bytes in flight (< 5 KB) and that the intermediate nodes can completely cache this amount of data. They have found out that for packet loss rates below 10 %, the combined caching and NACK mechanism is more efficient than a reliable link layer approach based on ARQ due to the overhead by link level acknowledgements. On the other hand, processing NACK messages end to end only is extremely inefficient for packet loss rate above 10 %. These findings are consistent with design principles of DTC and TSS. Similar as RMST, DTC and TSS use caching and local retransmissions by intermediate nodes without introducing pure link level ARQ, but both rely on information from existing protocols on the link or transport layer only.

Pump Slowly Fetch Quickly (PSFQ) [10] is a reliable transport protocol for re-tasking and re-programming of sensor nodes. The main PSFQ idea is to pump data rather slowly towards the receiving sensor nodes, but to recover missing data locally from intermediate nodes. The pump operation aims to support quick forwarding in case of no errors and behaves like a store and forward approach in situations with a high number of errors. The pump operation is based on broadcasting packets hop-by-hop from source to destination. Segment numbers are used to discover duplicates. Nodes receiving a packet add random delays before re-broadcasting in order to avoid collisions. While packet forwarding based on re-broadcasting have significant advantages in dynamic environments such as mobile ad-hoc networks and networks with unsynchronized sleep cycles [11], simulation experiments have shown that already a low number of packet losses due to congestion or bit errors can cause a significant number of duplicated packet. Duplicates, however, cause unnecessary packet reception, processing and transmissions, which should be avoided for energy-efficient sensor networks. The fetch operation is based on proactively requesting retransmissions from neighbour nodes using NACK messages and applies the concept of loss aggregation. In case that the last message of a packet sequence is lost, a fetch operation is triggered by a timeout. Multiple lost messages can be recovered in a single fetch operation. In addition to NACK messages, PSFQ introduces report messages for reporting the reception status at the destination to the source. The backpressure mechanism of TSS has a similar effect: Packet forwarding will be slowed down as soon as errors are detected by the intermediate nodes. A TSS node stops forwarding a packet, if previous packets have not been forwarded by successor nodes. In contrast to PSFQ, which includes a specific routing and forwarding mechanism, TSS is independent of the routing protocol.

Event-to-Sink Reliable Transport (ESRT) [12] aims to support reliable sensor data transport in wireless networks. It includes congestion control and mechanisms to achieve reliability. The reliability is controlled by adapting a rate at which the sink sends state reports back to the source. The frequency of the reports depends on the observed and desired reliability as well as the needs from congestion control. As in the case of PSFQ, a special protocol has been proposed, while no transport protocol extensions are required in TSS.

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 towards the source using backpressure signals (open-loop). Nodes receiving backpressure signals throttle down their transmission. In addition, a closed-loop mechanism operates on a longer time-scale. Based on acknowledgements received from the sink, sources regulate themselves. Lost acknowledgements result in reducing the rate at the source. Again, in contrast to TSS, new signalling messages need to be introduced into CODA.

Congestion control is very important in wireless sensor networks, because overloading a wireless network by too many transmissions can increase the collision probability. Collisions lead to packet losses and unnecessary retransmissions, which make sensor network operation energy-inefficient. TCP congestion control limits the maximum window size according to the slow start congestion control algorithm. However, it even might make sense to further limit the window dependent on the number of intermediate hops in a wireless multi-hop network, because the optimal window size in terms of throughput might be below the window size of standard TCP [14]. For example, it has been proposed to limit the maximum congestion window size to (number of hops) / 4 in a chain of nodes that are 200 m away from each other and have 250 m transmission range and 550 m interference range. This result shows that it might be beneficial to limit the TCP congestion window in wireless multi-hop networks such as a sensor network. The backpressure mechanism used in TSS has a similar effect and in our simulation limited the maximum congestion window size in a chain of 10 hops to three.

While several approaches perform packet caching for local retransmissions in case of packet loss due to congestion or lossy channels, other related works apply caching to recover from more serious errors such as disconnection of networks or route breaks. The design of a smart link layer is proposed in [15]. Packets might be re-received after a disconnection in order to re-trigger TCP after a longer disconnection period by putting TCP packets such as acknowledgements again into the input TCP queue. Re-sending packets to the peer can also

facilitate restart of TCP in such a case. The proposed mechanisms are rather orthogonal to the concepts proposed in this paper.
In [16] it is also proposed to hold copies of forwarded packets in a cache. When a downstream node encounters an error with packet forwarding, a route error message might be sent to the upstream node. The cached packet can then be retransmitted possibly on multiple alternative routes in order to repair the route break. TCP with BUffering capability and Sequence information (TCP-BUS) [17] proposes to buffer packets during route disconnection and re-establishment. After a route becomes available again, buffered packets are retransmitted by intermediate nodes. Special control messages are used to indicate route breaks and re-establishments. TCP can adapt its behaviour dependent on the knowledge that packets have been lost for other reasons than congestion. While DTC and TSS do not explicitly focus on route breaks, such failures can also be supported by both approaches.

4. Distributed TCP Caching

4.1. Overview

The key idea of Distributed TCP Caching (DTC) is to avoid energy-costly end-to-end retransmissions by caching TCP segments inside the network and retransmitting segments locally, i.e. from the intermediate sensor nodes' caches, when packet loss occurs. Ideally, each node would cache all segments and perform the retransmission exactly from the last node that has transmitted a segment before it has been lost. However, due to the constrained resources of the sensor nodes, we assume that each node can only cache one segment. Nodes take extra care to cache segments presumably not received by the next node. DTC is only implemented in the intermediate sensor nodes and does not require any changes on the TCP endpoints. A sensor node acting as the receiver may make use of the following standard TCP features: The receiver announces a small maximum segment size in order to avoid large TCP segments exceeding the capacity of the sensor nodes. Further, the receiver announces a small window size constraining the number of segments in flight.

4.2. Protocol Mechanisms

Caching

Due to the memory constraints of the sensor nodes, it is vital to the performance of DTC to find an appropriate way for nodes to select which segments to cache. A desirable outcome of this selection is that all segments that are currently in flight are cached and extra care is taken to cache segments that are likely to be dropped along the path towards the receiver. To achieve this, nodes cache TCP segments with the highest segment number seen with a certain probability. This assures that some older segments can be cached in the network as well. In order to detect packet loss at the next hop, we use feedback from a link layer that deploys positive acknowledgments. Our design also works with overhearing, i.e. when a node overhears that its successor transmits the packet. A TCP segment that is forwarded but for which no link layer acknowledgment has been received may have been lost in transit. Therefore, the segment is *locked* in the cache indicating that it should not be overwritten by a TCP segment with a higher sequence number. A locked segment is removed from the cache only when a TCP ACK that acknowledges the cached segment is received, or when the segment times out.

Packet Loss Detection and Local Retransmissions

To avoid end-to-end retransmissions, DTC needs to respond faster to packet loss than regular TCP. DTC relies mainly on timeouts to detect packet loss. Every node participating in DTC maintains a soft TCP state for connections that pass through the node. We assume symmetric and relatively stable routes, and therefore the nodes can estimate the delays between the node and the connection end-points. Each node measures the round-trip time (rtt) to the receiver and adapts a retransmission timeout to 1.5 * rtt. This ensures that the retransmission timeout values are smaller for nodes close to the destination and higher for nodes close to the source. Since the rtt values experienced by the nodes are lower than those estimated by the TCP end-points, the intermediate nodes are able to perform retransmissions earlier than the TCP end-points. DTC nodes set a timer for a local retransmission when they lock a segment in the cache. Simulations have shown that the other standard TCP mechanism to detect packet loss, duplicate acknowledgements, cannot contribute significantly to the performance of DTC.

Selective Acknowledgements

DTC uses the TCP SACK option to both detect packet loss and as a signalling mechanism between DTC nodes. DTC uses the latter to inform other nodes about the segments locked in the cache. On reception of a TCP ACK with an acknowledgement number smaller than the sequence number of its cached segment a node performs the following actions:

- If a node's cached segment's sequence number (`cached`)is not in the SACK block, the node retransmits the cached segment. Before transmitting the TCP ACK towards the sender, the node adds `cached` to the SACK block. Moreover, if `cached` fills all gaps, i.e. with `cached` all segment numbers up to the highest in the SACK block are acknowledged, the node can drop the acknowledgement. Note that the node should not generate a new ACK acknowledging all the segments in the SACK blocks since the receiver is allowed to discard a previously SACKed segment.
- The node can clear its cache if the cached segment's sequence number is in the SACK block since this means that either the receiver has received the corresponding segment or that the segment is cached and locked by a node closer to the receiver.

Note that even if the sender does not support SACK, the base station or the first node in the sensor network might add and remove the SACK options to enable SACK signalling for DTC.

Local Regeneration of TCP Acknowledgements

While DTC caches TCP data segments, it does not cache and retransmit TCP ACKs. DTC uses a simple local regeneration of TCP acknowledgements. When an intermediate node sees a TCP data segment, for which it has already received and forwarded a TCP ACK, the node assumes that the TCP ACK has been lost. Therefore, it does not forward the data segment but instead locally regenerates a TCP ACK. Note that this can be done without any buffering or caching of the original TCP ACK.

Example Operation

Figure 1 shows an example of DTC using SACK as a signalling mechanism. In this example, a TCP sender transmits three TCP segments. Segment 1 is cached by node 5 before it is dropped in the network. Since node 5 does not receive a link layer ACK, it locks segment 1 in the cache. Similar, segment 2 is cached and locked by node 7. When receiving segment 3, the receiver sends an acknowledgment ACK 1 with a SACK block for segment 3. On reception of the ACK segment, node 7 retransmits segment 2, adds a SACK block for segment 2 and forwards the acknowledgment. Eventually node 5 receives that acknowledgment and retransmits segment 1. Since all the gaps are filled now, node 5 drops the acknowledgment. Having received both segment 1 and segment 2, the receiver transmits ACK 4. Due to space limitations the TCP ACK sent by the receiver on reception of the retransmitted segment 2 is not shown Figure 1.

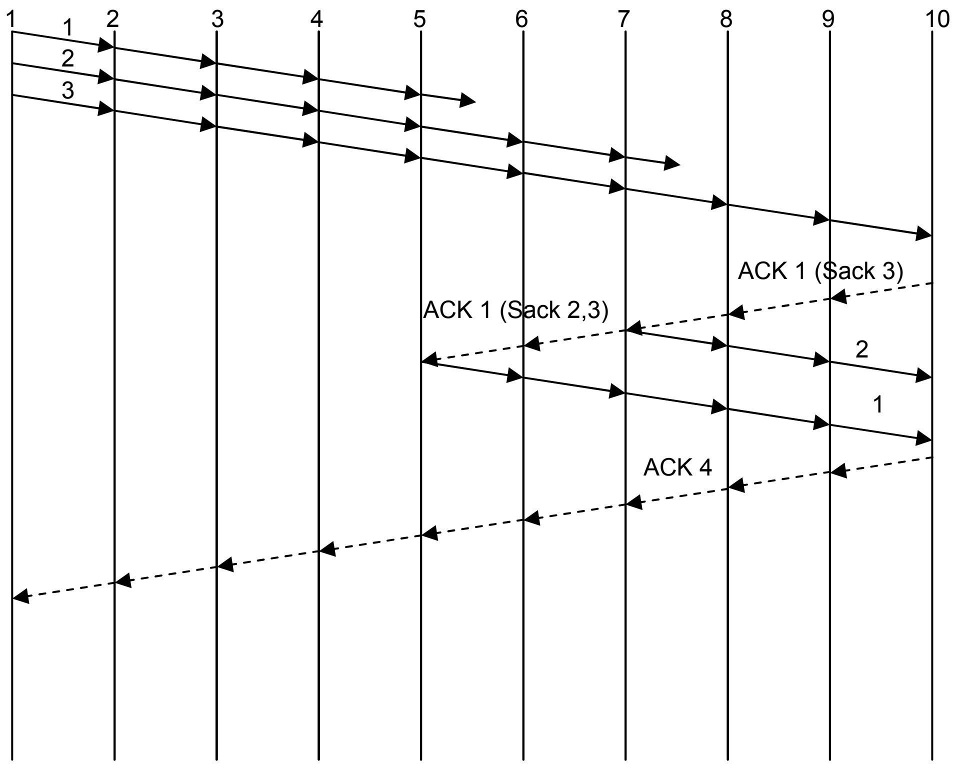

Figure 1: DTC using SACK as signalling mechanism

5. TCP Support for Sensor Nodes

5.1. Overview

TCP Support for Sensor nodes (TSS) is a layer between TCP and the routing layer to be implemented in a communication protocol stack of sensor nodes and has the goal to support energy-efficient operation of sensor nodes. TSS should ideally be implemented in TCP sensor nodes with senders and receivers as well as in intermediate sensor nodes that relay TCP segments and acknowledgements of a TCP connection. As DTC, TSS requires to store state information for each TCP connection. The state information is less than 20 integer values and contains mainly sequence and acknowledgement numbers, and rtt estimates. TSS tries to reduce the number of transmissions by a number of concepts and mechanisms:

- Caching of packets that might not have been received by the successor node (next node) based on overhearing and TCP acknowledgement spoofing
- Local retransmission of TCP segments based on round trip time estimation
- TCP acknowledgment regeneration and recovery based on forwarding delay estimation and overhearing
- A backpressure mechanism avoids that a node forwards a packet if the successor node might not have received all previous packets.

The mechanisms are completely based on TCP segments and TCP acknowledgments. They do not require any link level acknowledgments. Moreover, the mechanisms do not assume to have a CMSA/CA like scheme based on MAC level acknowledgements. TCP segments and acknowledgements are the only packets that are needed. This approach further reduces the amount of transmissions and can therefore be used on top of any kind of sensor network MAC layer. TSS also ensures that packets arrive at the destination in sequence. This avoids any re-sequencing buffer and selective acknowledgement / retransmission extensions in TCP.

5.2. Protocol Mechanisms

Caching

Similar as DTC, TSS allows to cache TCP segments in intermediate nodes. However, unlike in DTC, the decision to cache a TCP segment is completely deterministic. A node always caches that TCP segment with the data containing the first byte that has not yet been acknowledged or forwarded by the successor node. The packet is cached until it is sure that the successor node towards the destination has received the segment. A node knows that the successor has received a segment when it detects that the successor has forwarded the segment or when it spoofs a TCP acknowledgment that has been sent from the receiver toward the sender of the TCP 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 segments. A packet that is known to be received by the successor node will be removed from the cache. As in DTC the cache can hold a single packet. However, the mechanisms selected for TSS require another packet buffer (simply called buffer hereafter) for temporarily storing packets that are waiting to be forwarded to the successor node.

Local Retransmissions of TCP 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 of a TCP segment or acknowledgement. Retransmissions are mainly triggered by timeouts, which requires careful setting of timeout values. As for DTC the retransmission timeout is set to 1.5 * rtt. Simulations showed that a retransmission timeout of 2 * rtt performs slightly worse. A retransmission timeout of 1.5 * rtt allows to react somewhat earlier on packet loss and allows to repair even multiple packet losses before an end-to-end timeout is triggered.
The timeout values that are increasing from destination to source also meet the needs of the backpressure mechanism described below. The maximum number of local retransmissions has been limited to four. One might argue that forcing sensor nodes to overhear packets does not support energy efficient operation. On the other hand, 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. Another problem might be that a forwarding node transmits with reduced power such that the previous node can not overhear the forwarded packet. However, it might be much more efficient to adapt the transmission power to both the next and the previous hop and perform a single TCP segment transmission than to transmit both TCP segment to the next hop and link level acknowledgement to the previous hop with individually adapted power.
Moreover, it might happen that a node's retransmission timeout expires and the node retransmits a TCP data segment although that one has been received and forwarded by the successor. This might happen, if the previous node receives an overheard packet header with an error and drops that implicit acknowledgement. Then, the already correctly forwarded TCP segment should not be forwarded again. Forwarding should be prevented by a filter that filters out all segments that have been forwarded previously. Of course, end-to-end retransmissions should not be filtered, because end-to-end retransmissions might be needed to recover from severe problems such as route breaks, which a forwarding node might not be aware of. Retransmitted TCP segments can be uniquely identified by the source address and the IP identification field.

Regeneration and Recovery of TCP Acknowledgments

TCP acknowledgements are extremely important for TSS, since several mechanisms such as round-trip-time estimation, retransmission and caching depend on it. Experiments have shown that loss of acknowledgements may have a severe impact on the amount of TCP segment transmissions. TSS deploys two mechanisms for retransmissions of TCP acknowledgements that help to decrease the number of TCP segment transmissions significantly: A local regeneration mechanism as deployed by DTC and an aggressive TCP acknowledgement recovery mechanism, which recovers TCP acknowledgments if their forwarding by the successor node has not been discovered. Since TCP acknowledgments should usually be forwarded without significant delay towards the sender of TCP segments, each node measures the time between the TCP acknowledgment transmission and the overhearing of the TCP acknowledgment transmission from the successor node towards the TCP segment sender. Similar as for the rtt estimation we use an exponential averaging scheme and set the TCP acknowledgment timeout to the double average value. After timeout expiration, a TCP acknowledgment is recovered using the highest acknowledgment number. Again, this does not require to cache acknowledgments, but the acknowledgments can be recovered using state information.

Backpressure Mechanism

If the successor of a node has not forwarded all the 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 TCP segment with bit error needs to be recovered first. If a node would continue with packet forwarding in such a case, the risk that it performs unnecessary transmissions would be rather high. In a congestion situation, a forwarded segment might easily get lost and the same is true in case of a lost packet due to bit errors, because in such a situation all caches on subsequent nodes are occupied and the transmission of a new packet would not be protected. For that reason, in TSS a node stops any forwarding of subsequent packets until it knows that all earlier packets have been received and forwarded by its successor. If packet forwarding stops at some point, it will affect all the other nodes in the chain behind the stopping node. This means that all the other nodes will also stop their transmissions until progress is detected at their respective successor nodes. In the case of a lost packet (due to congestion or bit errors), the packet loss should be recovered by the node that sent the packet at last. In that case, we also have to avoid that retransmissions are triggered by nodes behind the recovering node, i.e. the nodes closer to the sender. We therefore have to increase the retransmission timeouts at the nodes closer to the sender. For that reason, the mechanism ensuring that the retransmission timeouts increase along the nodes from the receiver to the sender as explained in subsection “Local Retransmissions of TCP Segments” perfectly fits to the backpressure mechanism.

5.3. Example Operation

Figure 2 illustrates the operation of TSS. The first segment is forwarded without error from the sender to the receiver, while the second segment is lost between nodes 6 and 7. We assume here that node 5 overhears the forwarded packet from 6 to 7 and that node 5 therefore assumes that node 6 has successfully forwarded the segment to 7. This situation can easily occur, if node 6 is closer to node 5 than to node 7 or if the transmission from 6 to 7 is disturbed by another transmission such as from 10 to 9, while the latter one does not disturb the transmission from 6 to 5. In our example, node 6 caches the second segment and will timeout. In order to avoid that node 6 has to drop the third segment, we have to provide an additional packet buffer for buffering the third segment. This segment will not be forwarded and therefore, node 5 will stop forwarding subsequent packets. Assuming the nodes have measured the rtt as described above, node 6 times out before node 5 and retransmits the second segment to node 7. Node 5 will overhear that transmission and continue with packet forwarding, i.e. with forwarding packet 4. In general, the timeouts (resulting from the measured rtts) at nodes closer to the TCP receiver must be smaller than the timeouts at nodes closer to the sender. If we assume the minimum round trip measured for the first segment, we see that node 6 times out before node 5. However, in this special case, the timeout for the fourth packet expires too soon at node 1, but typically the average rtt values will be higher at nodes far from the receiver, since retransmissions will also contribute to the rtt calculation. More severe problems result from multiple packet losses. For example, if in our scenario the retransmission of the second segment by node 6 would be unsuccessful again, nodes 5 would time out too early and retransmit unnecessarily.

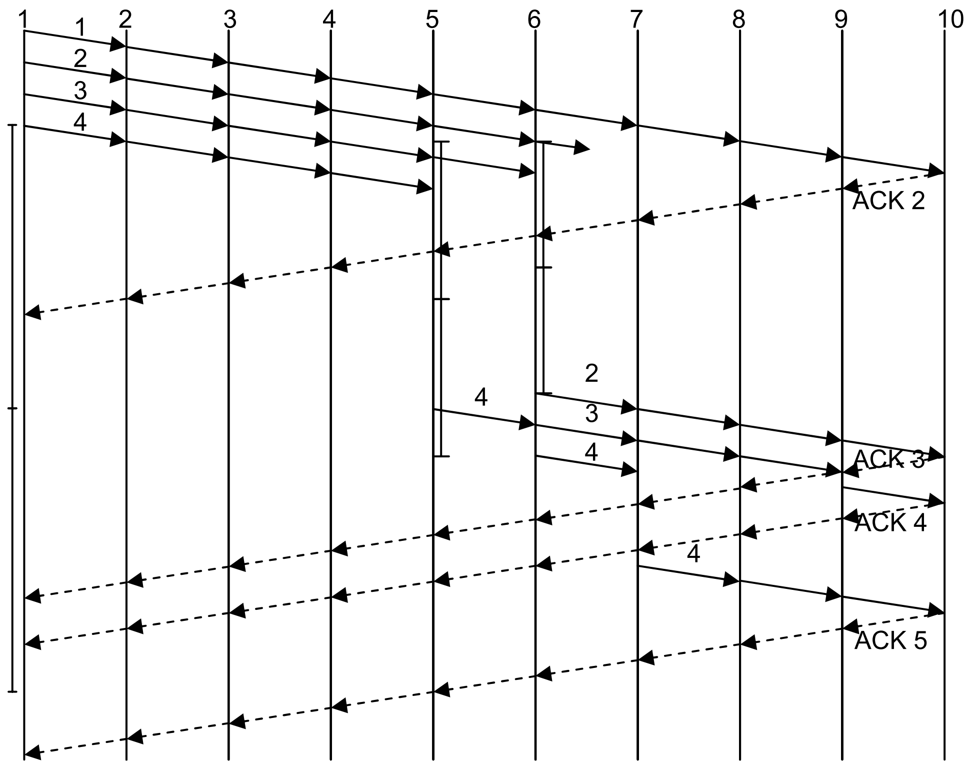

Figure 2: TSS operation in case of a lost TCP segment

In another scenario shown in Figure 3, an error might occur with the second segment between nodes 6 and 7. The segment is correctly received by node 6 and node 7 forwards it correctly to node 8. However, the transmission from node 7 to node 8 is disturbed at node 6 and node 6 does not even receive the packet header of the third packet forwarded from node 7 to node 8. Node 6 therefore assumes that the second packet has not been received by node 7 and stops the transmission of new packets. However, after a while the TCP acknowledgment for segment 2 arrives at node 6 and node 6 can then continue to forward the third segment.

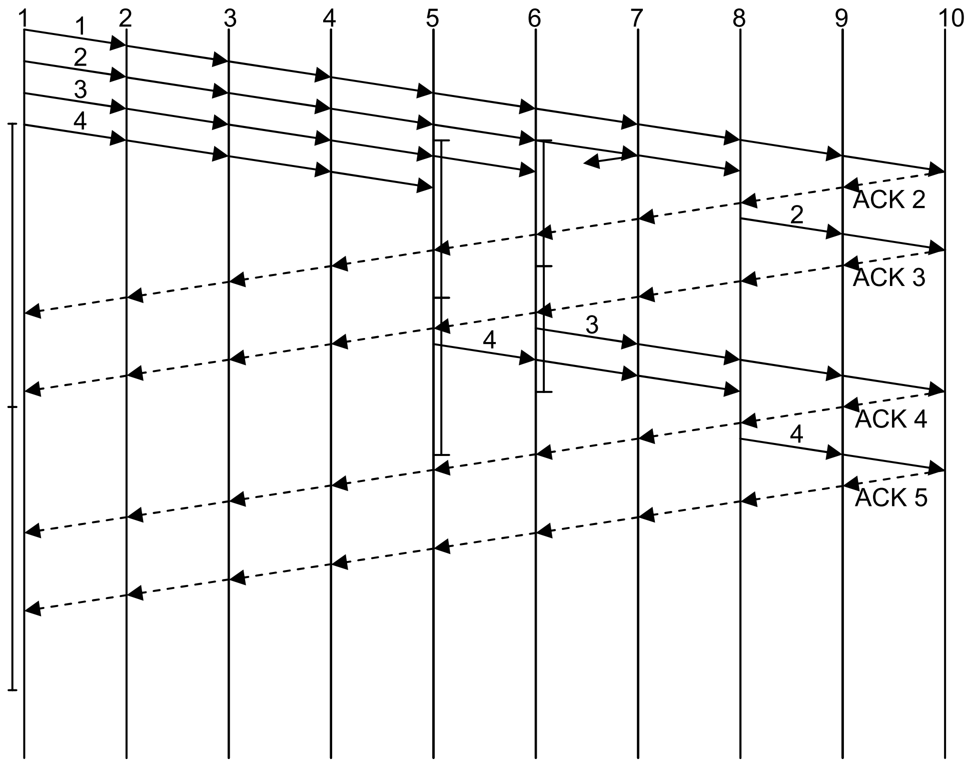

Figure 3: TSS operation in case of an overhearing error

5.4. Comparison with DTC

While both DTC and TSS allow intermediate sensor nodes to cache packets and locally retransmit packets in case of errors, there are some differences between the approaches: In DTC each segment is forwarded immediately while TSS's backpressure mechanism keeps cached segments until it knows that the previous segments have been received by the next hop node. Furthermore, DTC requires one buffer for storing / caching TCP segments, while TSS needs two buffers. While DTC uses selective acknowledgments and retransmissions, TSS does not make use of this option and hence requires less re-sequencing buffers at the receiver.

6. Performance Evaluation

6.1. Simulation Scenarios and Parameters

Both, DTC and TSS have been evaluated using simulations with Omnet++ [18]. Figure 4 shows the simulation scenarios used for our evaluation. In the case of DTC, we have a TCP sender (source) and a TCP receiver (destination) implementation that exchange 500 TCP segments with a size of 800 bits including TCP/IP and MAC header. Ten intermediate nodes are interconnected in a chain of nodes. In case of TSS, we have a TSS implementation on each node. These are somewhat different for intermediate nodes, TCP source and TCP destination. However, the TCP implementation used at the source and the destination are identical for the DTC and TSS simulation experiments.
We assume that we do not have any interference between the nodes, which can be achieved using a time division multiplexing scheme on the MAC layer. The selected wireless network bit rate was 19.2 kbps. The TCP acknowledgement packet size was 352 bits for DTC. For TSS we assume a somewhat larger MAC header that includes 32 bit MAC addresses of the next, the sending and the previous node. This results in a size of 416 bits for implicit acknowledgements. Overhearing does not really require a MAC header with these three addresses, but makes the implementation much simpler. The TSS modules have been interconnected via channels of a certain uniformly distributed bit error rate. The simulation

experiments have been performed for packet error rates (PER) of 0, 5, 10, and 15 %. The bit error rate (BER) for the packet error rates can be calculated by $BER = 1 - \sqrt[n]{1 - PER}$, n: bits per packet. The corresponding bit error rates for n = 800 are 0, 0.000064114, 0.000131697, and 0.000203128. In the case of TSS, we assumed that a node considered an overheard TCP segment as correctly received, if the TCP/IP and MAC header has been received without error. For each experiment we calculated the average values over 30 simulation runs.

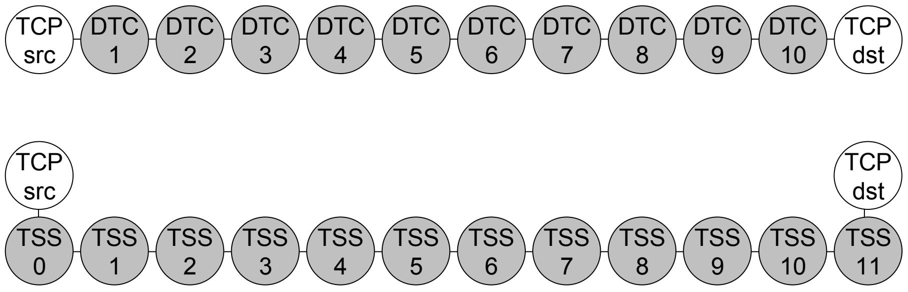

Figure 4: Simulation scenarios

6.2. Performance Results

Figure 5 compares the total number of transmissions for TCP, DTC, and TSS. It shows that for no and low (0 and 5 %) packet error rates (PER), DTC and TSS have a similar number of total packets, while the total number of packets is somewhat lower for TSS in case of high (10 and 15 %) packet error rates. It is obvious that for higher packet error rates, TCP without any support in intermediate nodes has extremely poor performance. This is in particular due to the very high number of end to end retransmissions, which are shown in Figure 6.

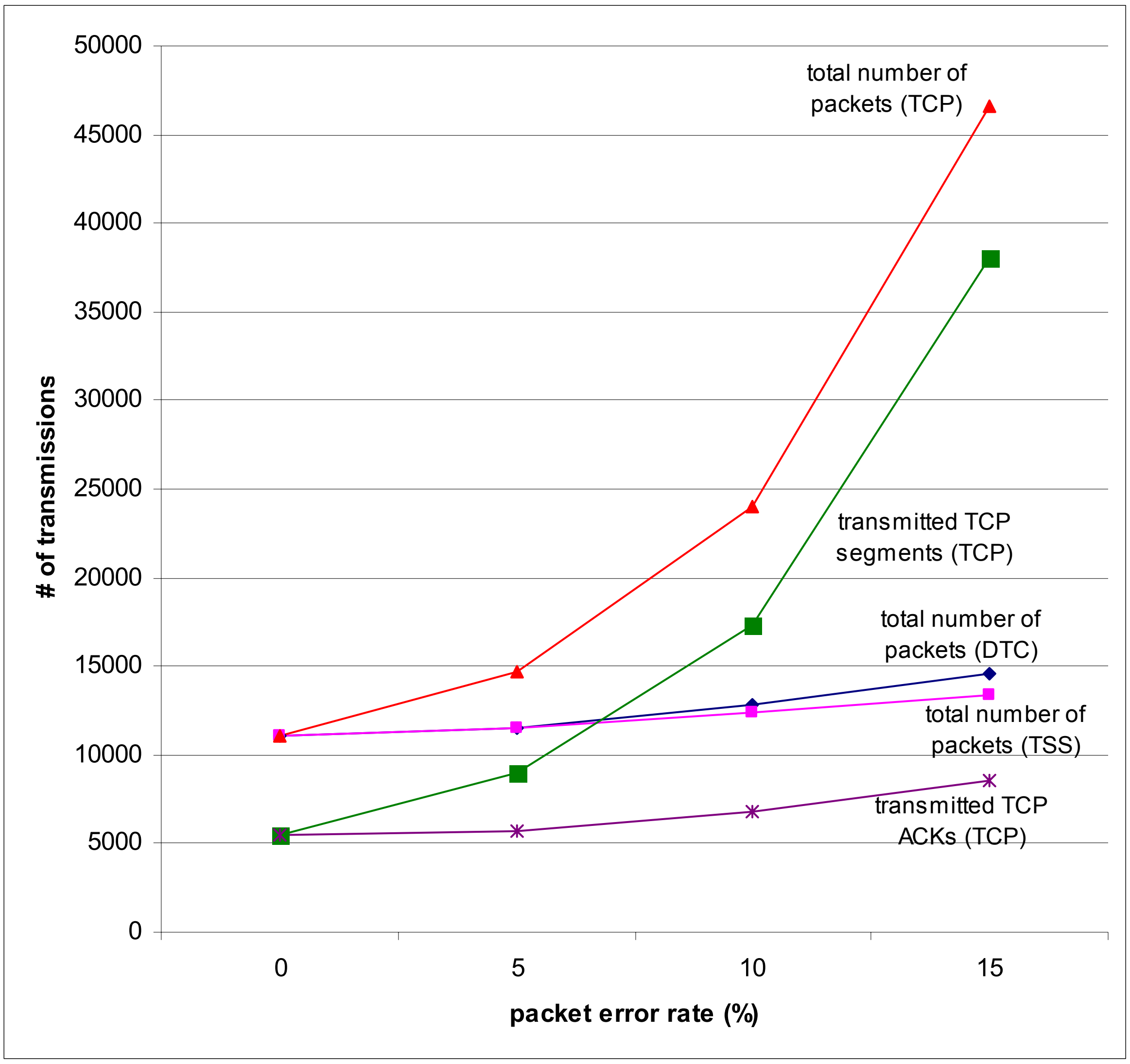

Figure 5: Number of TCP segment and acknowledgement transmissions for DTC and TSS

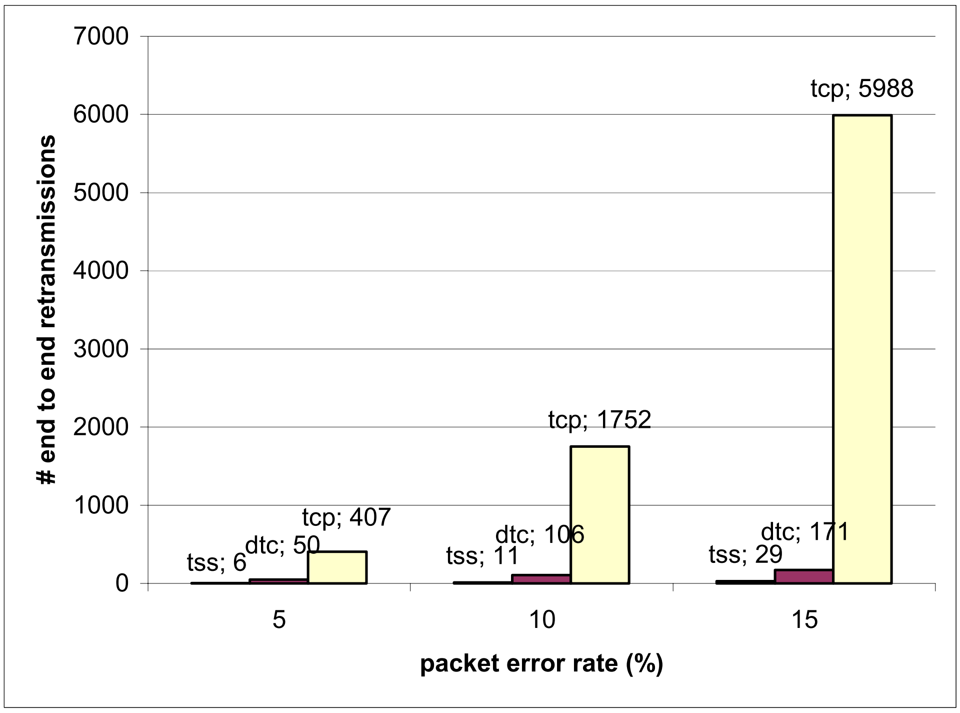

Figure 6: Number of end to end retransmissions for TSS, DTC, and TCP

Figure 7 analyses the transmissions of DTC and TSS in more detail: It can be seen that the number of TCP data segments is always lower for TSS, while the number of acknowledgements is always higher. This results from the aggressive acknowledgment recovery scheme implemented in TSS, while DTC does not implement such a scheme. Due to the cumulative nature of TCP ACKs (an ACK acknowledges *all* segments up to the acknowledgment number minus one), the number TCP ACKs DTC transmits can become lower when packet loss occurs than in the error-free case. The dashed line shows the minimum number of TCP segment transmissions, assuming that each packet error can be recovered by a single local retransmission. This minimum number can be calculated by

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

The minimum number of acknowledgments is somewhat difficult to determine, since it depends on the acknowledgement retransmission scheme as discussed above.

Not shown in the figures is the performance of TCP running on top of a reliable link layer (relTCP) which we have also simulated. The results do not depend very much on the maximum number of retransmissions and buffer sizes if these are between three and five, but performance degrades for a buffer size of two. For a buffer size of three, at maximum three link layer retransmissions and 10% packet error rate, the number of end-to-end retransmissions for relTCP (72) is higher than for TSS but lower than for DTC. While the number of transmitted TCP data segments of relTCP (7397) is about the same as for DTC and thus higher as for TSS, the number of acknowledgements is higher (6609) than for both DTC and TSS since relTCP is not able to exploit the cumulative nature of TCP acknowledgements.

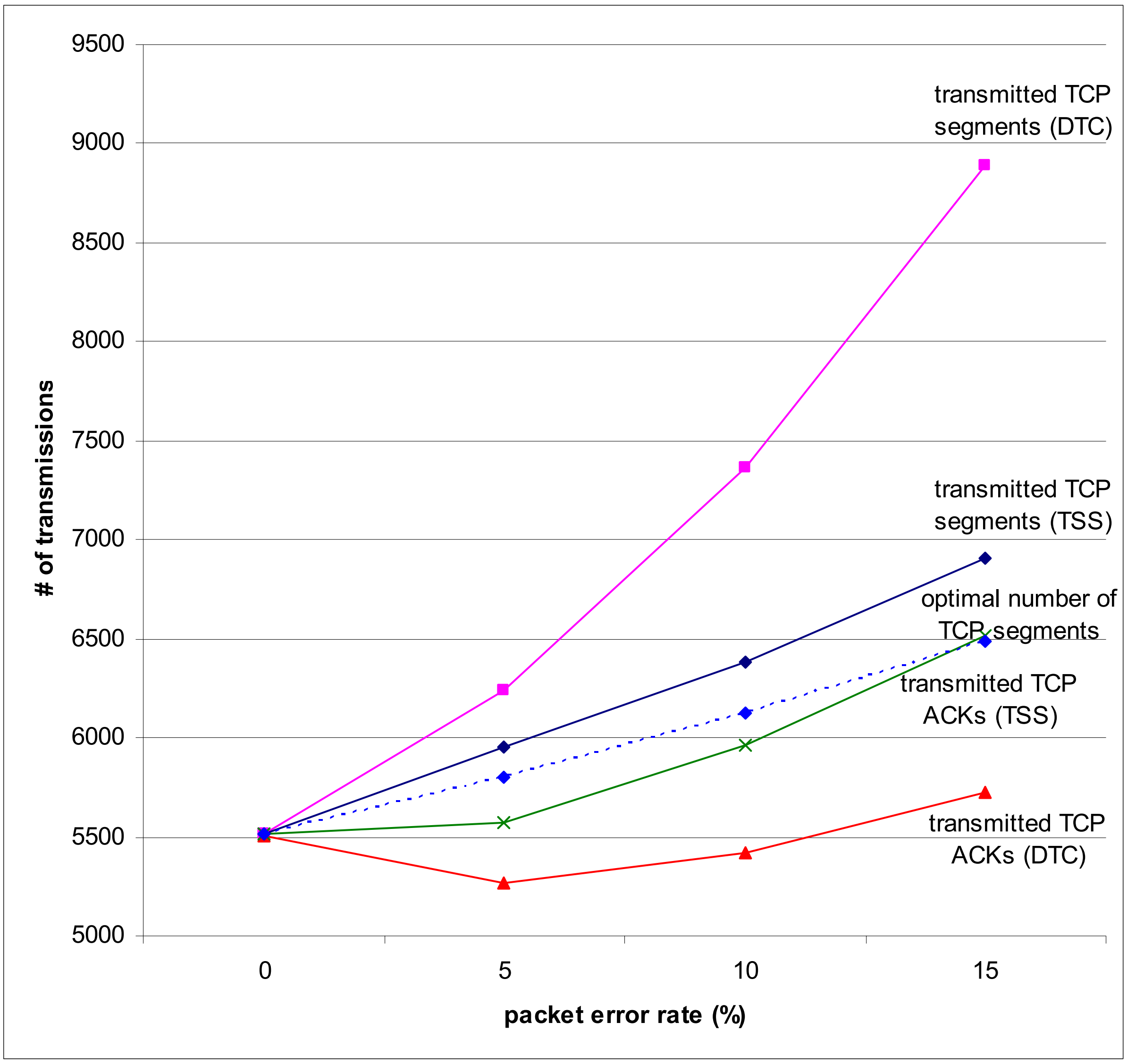

Figure 7: Transmitted TCP segments and TCP acknowledgements for DTC and TSS

7. Conclusions

TCP support in wireless sensor networks is desirable to allow direct communication of sensor nodes with other systems for various purposes such as configuration, re-programming or management. Recent work showed that TCP/IP can be implemented on sensor nodes with limited processing power and memory [4]. In this paper we show that even in scenarios with high error rates, TCP can be used and implemented in an energy-efficient way. This requires some protocol support in intermediate nodes that are able to store TCP segments for possible local retransmissions. Both concepts presented in this paper drastically reduce the number of TCP segment transmissions that are needed to transfer a certain amount of data across a wireless sensor network with relatively high bit / packet error rates. Both approaches have different requirements on TCP protocol options (selective retransmissions vs. go back n) and buffer requirements (1 vs. 2 packet buffers). The different types of link level support (link level acknowledgments vs. overhearing / implicit acknowledgements) are rather implementation issues and can be easily used by either mechanism. Future work might analyse the performance in further network scenarios, integrate scheduling mechanisms for sleep cycles and consider real implementation on available sensor node hardware. Further reduction of transmissions might be achieved by delayed acknowledgements and combining data and acknowledgement transmissions.

References

1. A. Dunkels, T. Voigt, H. Ritter, and J. Alonso: Distributed TCP Caching for Wireless Sensor Networks, Annual Mediterranean Ad Hoc Networking Workshop 2004, Bodrum, Turkey, June 2004.
2. A. Dunkels, T. Voigt and J. Alonso: Making TCP/IP Viable for Wireless Sensor Networks, Work-in-Progress Session of the 1st European Workshop on Wireless Sensor Networks (EWSN 2004), Berlin, Germany, January 2004.
3. A. Dunkels, T. Voigt, N. Bergman, and M. Jönsson: The Design and Implementation of an IP-based Sensor Network for Intrusion Monitoring, Swedish National Computer Networking Workshop, Karlstad, Sweden, November 2004.
4. A. Dunkels: Full TCP/IP for 8-bit Architectures, MOBISYS 2003, San Francisco, California, May 2003.
5. A. Dunkels: The uIP TCP/IP Stack for Embedded Microcontrollers, web page, visited 2004-11-21. `http://www.sics.se/~adam/uip/`
6. Scatterweb.net, web page, visited 2004-11-21, `http://www.scatterweb.net/`
7. D. Estrin, R. Govidan, J. Heidemann and S. Kumar: Next century challenges: scalable coordination in sensor networks, Mobile Computing and Networking, pp. 263-270, 1999.
8. H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz: Improving TCP/IP performance over wireless networks, International ACM Conference on Mobile Computing and Networking, MOBICOM 1995, pp. 2-11, November 1995.
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. 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.
11. M. Heissenbüttel, T. Braun, Th. Bernoulli, and M. Wälchli: BLR: Beacon-Less Routing Algorithm for Mobile Ad-Hoc Networks, Computer Communications Journal, Elsevier, Vol. 27, No. 11, July 2004, pp. 1076-1086.
12. Y. Sankarasubramanian, Ö. Ankan, I. F. Akyildiz: ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks, ACM MobiHoc, June 1-3, 2003, Anaheim.
13. C.-Y. Wan, S. Eisenman, A. Campbell: CODA: Congestion Detection and Avoidance in Sensor Networks, ACM SenSys 2003, November 3-5, 2003, Los Angeles, USA.
14. 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, March 30 - April 3, 2003, San Francisco.
15. J. Scott, G. Mapp: Link Layer-Based TCP Optimisation for Disconnecting Networks, ACM SIGCOMM Computer Communications Review, Vol. 33, No. 5, October 2003.
16. A. Valera, W. Seah, S. Rao: Cooperative Packet Caching and Shortest Multipath Routing in Mobile Ad hoc Networks, IEEE Infocom, March 30 - April 3, 2003, San Francisco.
17. D. Kim, C-K Toh and Y. Choi: TCP-BuS: Improving TCP Performance in Wireless Ad Hoc Networks, Journal of Communications and Networks, Vol. 3, No. 2, June 2001.
18. Omnet++: Discrete Event Simulation System, web page, visited 2004-11-21, `http://www.omnetpp.org`