TCP/IP communication in a WSN

Oliver Gasser
Betreuerin: Corinna Schmitt
Seminar Sensorknoten: Betrieb, Netze und Anwendungen SS2011
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: gasser@in.tum.de

ABSTRACT

In this paper, the current state of TCP/IP adaption in wireless sensor networks (WSNs) is surveyed. WSNs are becoming more and more ubiquitous for all kinds of monitoring applications and also for ad-hoc networking. TCP/IP on the other hand is the de-facto networking standard. In its pure form, however, the protocol suite does not perform well in a WSN in terms of energy consumption and other factors. Fortunately there are several approaches that try to adapt TCP/IP to sensor networks which will be presented in this paper.

Keywords

TCP, TCP/IP, WSN, Sensor Network, Caching, Energy Efficiency, Link Quality, MSS

1. INTRODUCTION

Wireless Sensor Networks are becoming ubiquitous not only in the scientific world but also more and more in your home (e.g. monitoring the temperature of rooms, alerting of an upcoming rain shower, etc.). Thus the communication between sensor networks and other networks is getting more important too. The de-facto networking standard protocol suite is TCP/IP. While it is possible to run TCP/IP on sensor nodes [7] it is not feasible (in terms of energy consumption) to run pure TCP/IP on them. The main factor for this is that TCP/IP was not designed to be used in wireless environments but rather in wired ones. The error rate in wireless environments is much higher and TCP cannot distinguish between losses due to congestion and losses due to bit errors. Another drawback of TCP/IP in WSNs are end-to-end retransmissions which are a huge energy waster if they are not kept to a minimum. That is why TCP/IP can and should be adapted to sensor networks. In this paper approaches to tailor TCP/IP to the characteristics of a WSN are presented.
The remainder of this paper is structured as follows. Section 2 explains in more detail the specifics of TCP/IP and wireless sensor networks. Section 3 surveys the challenges which are faced when making sensor nodes TCP/IP-ready and Section 4 lists and analyzes proposed approaches to solve the problems discussed in the previous section. Finally, this paper is concluded in Section 5, where the proposed approaches are rated.

2. TCP/IP IN A WSN

2.1 Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are becoming more and more omnipresent. Nodes in WSNs are mainly used for monitoring purposes such as environment monitoring (air pollution, temperature, humidity etc.) and machine monitoring (observing a machine's condition or health). Another application are mobile ad-hoc networks, where two computers (or similar devices) communicate with each other via WSN nodes without a router or any other kind of access point.
Due to their compact nature nodes in these networks have certain characteristics and constraints:

- Nodes should function unattendedly (except changing the battery every year or so).
- Energy consumption (CPU, sending, receiving) needs to be kept at a minimum to save battery power.
- Resources (CPU power, memory, energy) are scarce.
- The network must adapt to churn (i.e. nodes joining and leaving due to failure).
- A node's location can possibly change affecting the network's topology.

These characteristics on the other hand limit the nodes' universal applicability, i.e. sensor nodes are not able to run resource-consuming software as a generic PC would be able to. That is why the software for sensor needs to be specifically tailored to their abilities.

2.2 TCP/IP

For communication in the Internet and many other networks the Internet Protocol Suite is most often used. The two most important protocols from this family are the Transmission Control Protocol (TCP)[18] and the Internet Protocol (IP) [17]. Those two protocols are commonly referred to as TCP/IP and they are the de-facto standard protocols for Internet communication. TCP/IP is one reason why the Internet has become a big success story and is nearly ubiquitous today.
However, due to various design issues laid out in detail in Section 3, TCP/IP is not mainly used for WSN communication. Instead nodes run other WSN specific protocols. Despite that fact there are certain scenarios where it would be preferable to use TCP/IP in a WSN. The main gain of

running TCP/IP on sensor nodes is the ability to easily communicate from any other TCP/IP compatible device with any single node. This could be used to download gathered sensor data, reconfigure the node without directly attaching it to a PC or updating the software running on the node "over the air". Another scenario could be to use the WSN as an ad-hoc network to connect two TCP/IP devices which are otherwise unable to communicate with each other as they are not directly connected and have no access to the internet.

Internet Protocol

The Internet Protocol is an unreliable, connectionless layer 3 (i.e. network layer) protocol. That means that no connetion setup is performed and it is not guaranteed that packets arrive at the destination. The Internet Protocol exists in two versions: IPv4 and IPv6 [6]. Version 4 is predominantly used as of 2011, but the adoption of IPv6 is continuously increasing. IPv6 was primarily introduced to solve the problem of the exhaustion of IPv4 addresses. Since IPv4 addresses are 32 bits long therer are around 4 billion possible addresses. IPv6 on the other side features 128 bit long addresses which makes a total of 10^{38} different addresses. This increase in addresses, however, did not come for free: The IPv6 header is double the size of an IPv4 header, without extensions or options: 40 bytes vs. 20 bytes. In wireless sensor networks where most of the time only a few bytes are transferred and the link layer segment size is limited to a little more than 100 bytes, the 40 bytes of the IPv6 header alone would be too great of an overhead. Fortunately compression algorithms can reduce the IPv6 header's size to about 20 bytes [13]. Should the packet still not fit in a link layer frame it can be fragmented using the LoWPAN protocol layer [14].

Transmission Control Protocol

The Transmission Control Protocol is a reliable, connection-oriented protocol, in contrast to IP. It operates on layer 4 (transport layer) and establishes a connection between two TCP endpoints. Once the connection is established it needs to be managed. This management effort consists of making sure that all TCP segments are reliably transmitted from source to destination (retransmission on loss), detecting transmission errors, potential reordering of packets before delivering them to the destination process, detecting and reacting to network congestion, etc. Although these operations work very well on wired information systems and on wireless systems with no energy scarcitiy (e.g. home WiFi network), they do not work well in WSNs. The reasons for that are the sensor nodes' limited resources and the high error rates in wireless networks. An end-to-end retransmission after a bit error wastes too much energy since the packet has to be sent and received over the whole path again. TCP's reaction to packet loss is not optimal as it is interpreted as network congestion and the sending rate is reduced. This are just two of the problems, which will be discussed in the following section, of operating TCP in a WSN.

3. ISSUES OF ADOPTING TCP/IP IN A WSN

There are several issues which need to be solved, before TCP/IP is a viable protocol combination to be used in a WSN.
One issue is the header overhead. TCP and IP headers combined have a minimal size of 40 bytes: 20 bytes TCP header plus 20 bytes IPv4 header, without any additional options. If we look at the maximum size of link layer frames a significant part of the message is being occupied by header data. The IEEE 802.15.4 standard [11] for example limits the maximum size of link layer frames to 127 bytes. That leaves a mere 87 bytes of TCP payload even without taking the link layer header into account. The headers therefore occupy more than 30 % of the total maximum possible data which can be sent in one frame. The nodes' scarce energy resources are thusly not utilized in an optimal way. Additionally it should be noted that larger payloads can be fragmented into many packets. Fragmentation and reassembly, however, are themselves energy consuming processes.
The greatest hurdle which hinders TCP/IP from being widely adopted in wireless sensor networks is TCP's flow and congestion control mechanism. TCP is unable to differentiate between a lost segment due to congestion and a lost segment due to bit errors. Whenever a segment is lost, i.e. no acknowledgment is received for that particular segment and a timeout event is triggered, this loss is believed to be due to congestion in the network. Consequently the sending rate is reduced to avoid further segment losses. While this might be a good strategy in wired networks it certainly is not appropriate for wireless sensor networks, where bit error rates are orders of magnitude higher (up to double digit percentage package error rates [1]). Despite the fact that the loss occured due to bit errors the sending rate is reduced nevertheless. This leads to a less than ideal throughput.
Another issue with TCP in WSNs is TCP's connection management. Connections are maintained between the two endpoints of communication. If a packet gets lost in transit, a timeout occurs at the senders side or a duplicate acknowledgemt is received. The sender then retransmits the original packet to the receiver and hopes that it will go through this time. This behaviour is wasting the nodes' energy by forcing expensive retransmission on the whole path from sender to receiver.
In traditional IP networks unique IP addresses are assigned to each network interface based on the network's topology. This process of address assignment is either done manually or automatically (e.g. via DHCP). Assigning addresses this way is not very practical for sensor nodes. Furthermore sensor networks often prefer data-centric routing mechanisms instead of traditional address based routing [5]. A receiver simply announces its interest in a certain kind of data instead of nodes directly addressing a data sink.
Finally the energy, memory and CPU resources of a sensor node are very limited and it may be unfeasible to run a full TCP/IP stack on them. This, however, was proven to be possible [3, 7].

4. POSSIBLE SOLUTIONS

In this section possible solutions to the challenges faced in TCP/IP and WSN, which were discussed in Section 3, are presented.

4.1 Different scenarios and approaches

Dunkels et al. [8] listed three possible ways to connect sensor networks with TCP/IP networks: (1) via a proxy, (2) via DTN overlays or (3) implementing TCP/IP directly on

sensor nodes.
With the first method (see Figure 1) a proxy which resides on a gateway machine "translates" all TCP/IP packets directed to the sensor network to conform to the sensor network protocols. The exact opposite is done for packets sent from the WSN to the TCP/IP networks. Although this approach does not implement TCP/IP on the sensor nodes themselves it still makes it possible to access WSNs from a TCP/IP network. This approach has the advantage of being relatively simple to set up. Moreover security policies can easily be enforced on the gateway proxy. Drawbacks of this setup are the single point of failure nature of the proxy machine and the fact that different WSNs need different proxy implementations.

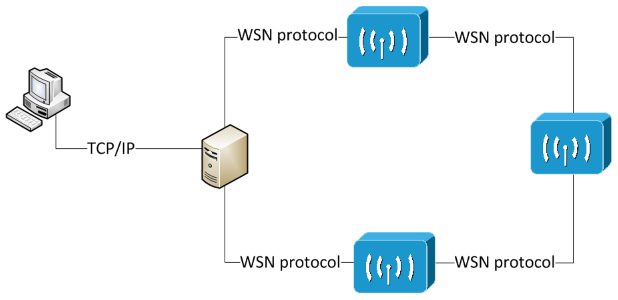

Figure 1: Proxy architecture.

Delay Tolerant Networks (DTNs) [10] are especially designed for environments with high bit error rates, long and changing delays, high churn and asymmetrical data connections. DTNs implement a network overlay and transmit messages (called "bundles") based on store-and-forward switching, i.e. a bundle is held available until the next hop confirms its reception. This avoids costly end-to-end retransmissions. A DTN is partitioned into regions and each region has one DTN gateway. This gateway is responsible for sending messages to other regions and to nodes in its own region. The DTN architecture can be seen as a generalization of the proxy approach.
To enable seamless integration between a TCP/IP network and a WSN the TCP/IP protocol stack should directly run on each sensor node. No gateways or other special nodes are needed in this approach. A visual representation can be seen in Figure 2. This approach, however, faces some issues which need to be addressed: Host-centric routing and addressing, large header overhead for TCP/IP, bad performance over links with a high bit error rate, end-to-end retransmissions. Possible solutions for these issues will be presented in the rest of this section.

Kuorilehto et al. [12] also list three different possibilities of integrating TCP/IP into WSN: (1) direct TCP, (2) proxy TCP and (3) native TCP.
The first approach is the same as the last method mentioned by Dunkels et al., Kuorilehto's second method corresponds to the first method by Dunkels et al. (see above).
Lastly the native TCP architecture transports TCP/IP traffic as a payload of the WSN protocols (see Figure 3). Their implementation called TUTWSN uses Time Division Multiple Access (TDMA, sending is only allowed in dedicated time slots) to avoid collisions which would lead to expensive re-

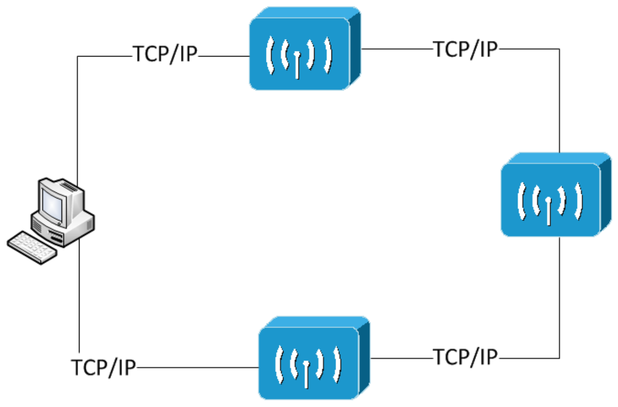

Figure 2: Direct TCP/IP implementation on sensor nodes.

transmissions. This setup allows for communications of two or more TCP/IP endpoints over a WSN with a gateway at each entry point. Direct TCP/IP communication with sensor nodes is not possible. In TUTWSN the sensor network is clustered, each cluster having one "cluster headnode". These headnodes control the communication within the cluster and cluster-to-cluster communication. The other nodes are called "subnodes" and cannot communicate directly with each other but rather via a cluster headnode. Each cluster communicates on a dedicated cluster channel which does not overlap with neighboring clusters. Routing decisions are made based on cost-gradients to a gateway. These costs could depend on the number of hops, remaining energy, number of associated nodes, power necessary to send a packet to the next hop, etc. Kuorilehto et al. suggest that the TDMA's idle periods could also be used for sending data which would result in a higher throughput. This, however, leads to more energy being consumed.

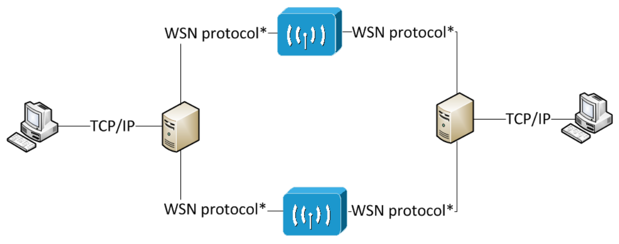

Figure 3: Native TCP/IP implementation transporting TCP/IP packets as payload (marked by a star).

4.2 Distributed TCP Caching

In this subsection and the following one, two similar approaches to make TCP in sensor networks more energy efficient [5] are presented: Distributed TCP Caching (DTC) [9] and TCP Support for Sensor networks (TSS) [4] which are both inspired by the Snoop protocol [2]. These approaches reduce the number of retransmissions by allowing intermediate nodes to cache packets and in case of packet loss do local retransmissions. The performance of both approaches is discussed in Section 4.4.

The basic idea in DTC is that intermediate nodes cache seg-

ments on their way and retransmit them locally to avoid costly end-to-end retransmissions. In the best case each node would cache all bypassing segments. In reality, however, this cannot be achieved as the nodes have only a very limited amount of storage. Thusly it is vital that the nodes very carefully select which segment to cache, hopefully caching those which will be lost. If a node receives a segment with the highest segment number seen until then it caches this segment with a certain probability. If the segment number is lower it is not cached. That behavior assures that not only the newest but also older segments are kept in the cache. If a TCP segment gets lost in transit to the next hop, the segment is locked in the cache. It will therefore not be overwritten by TCP segments with a higher sequence number. Link layer acknowledgements are used in order to detect segment loss. This could also be done with "overhearing", i.e. listening if the next hops forwards the segment. A cached segment is unlocked as soon as a TCP ACK acknowledging this very segment is received or when the segment times out. To avoid that a packet loss triggers a end-to-end retransmission DTC detects the loss before the TCP endpoint does. Each node maintains a soft state for passing TCP connections. This state saves the round-trip time (RTT) to the receiving node and sets the local retransmission timeout to $1.5 \cdot RTT$. The local retransmission timeout values are smaller for nodes close to the TCP receiver and get larger the nearer you come to the TCP sender. Since the local retransmission timeout is believed to be smaller than the TCP sender timeout local retransmissions kick in as a packet is lost and end-to-end retransmissions are avoided. Whenever a sensor node locks a segment in the cache the local retransmission timer is started. If it ends before the packet is unlocked, the segment is retransmitted.
To detect packet loss and for signaling purposes DTC uses the TCP selective acknowledgement (SACK) option [15]. If a node receives a TCP ACK there are two possibilities:

1. The acknowledged segment number is larger or equal to the cached segment, then the cache can be cleared.
2. The acknowledged segment number is smaller than the cached sequence number, then there are two subpossibilities:
   (a) If the cached segment's sequence number (`cached`) is not in the SACK block, the cached segment is retransmitted and `cached` is added to the SACK block. If all sequence numbers in the SACK block are contiguous the TCP ACK is dropped altogether, as all missing segments have been retransmitted and forwarding the ACK to the sender would trigger an unnecessary retransmission. If the SACK block is not contiguous the ACK is forwarded in the direction of the TCP sender.
   (b) If `cached` is in the SACK block, the node's cache can be cleared since the TCP receiver either already got this segment or it is locked in the cache by a node which is closer to the TCP receiver. Finally the TCP ACK is forwarded in direction to the TCP sender.

DTC has the ability to locally regenerate TCP acknowledgements without caching or otherwise storing them. If a node receives a TCP segment for which it already saw a TCP ACK the TCP segment is dropped and a TCP ACK is locally regenerated using the TCP connection's state information.

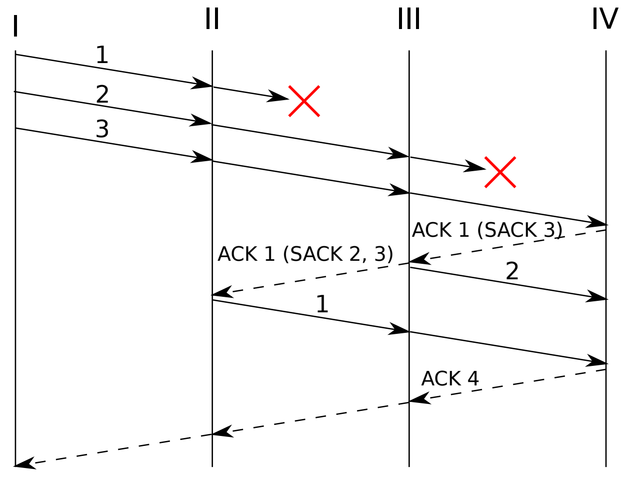

Figure 4: DTC example sending 3 packets over 4 nodes.

DTC example

The DTC example in Figure 4 shows the principle of the local recovery mechanism combined with the selective acknowledgement. The first packet gets lost in the network between node II and III but before that happens it gets cached by node II. Since node II does not receive a link level ACK from node III for the first packet, it gets locked in the cache. When the second packet arrives at node 2 it is not even considered to be cached as the cache is already occupied by a locked packet (the first packet). The second packet then gets cached at node III and is lost in transit from III to IV. With no link level ACK received, node III locks the second packet in its cache. The third packet arrives at the final node IV without being lost (it did not get cached at any node since all intermediate nodes had their caches locked). As node IV receives the third packet it sends the following TCP acknownledgement in direction to node I: `ACK 1 (SACK 3)`. Node IV is still waiting for the first packet but it already received the third packet. Upon reception of this TCP ACK message node III finds out that the ACK number is lower than its cached packet's and its cached packet's sequence number is not present in the SACK block either. In consequence node III restransmits the cached second packet, adds 2 to the TCP ACK and forwards it: `ACK 1 (SACK 2, 3)`. When node II receives the TCP ACK it does the same reasoning as node III did and restransmits its cached first packet. However, the TCP ACK is discarded as the SACK block now forms a contiguous sequence. As node IV receives the first packet it sends a TCP acknowledgement to node I indicating it is now waiting for the fourth packet: `ACK 4`.

4.3 TCP Support for Sensor Nodes

As DTC TCP Support for Sensor nodes tries to reduce the number of retransmissions by caching packets, retransmitting them locally, regenerating TCP acknowledgements and

a mechanism that avoids forwarding packets if the successor node has not received previously sent packets. Overhearing is used to notice if packets are received by the successor node, link level acknowledgements are not needed. As another benefit TCP segments arrive in sequence, hence no reordering or selective acknowledgements are required.
Unlike DTCs caching decision, TSS' is not based on a probability but completely deterministic. A segment which has not yet been forwarded or acknowledged by the successor is always cached. Nodes are listening to their neighbors' transmissions. If a node overhears that a packet has been forwarded by its successor it can remove this packet from the cache. The same holds if a TCP ACK sent from the TCP receiver to a neighboring node and acknowledging the cached packet is overheard. In addition to the buffer to store cached packets another packet buffer for temporal packet storage is needed. This buffer holds packets which have not yet been forwarded to the successor node but need to wait for the confirmation that the previously sent packet was received.
Local retransmissions are triggered by timeouts. The timeout is set to $1.5 \cdot RTT$ and is started after a package was completely sent. Then the node listens if the successor forwards the packet (overhearing). If it does not overhear anything before the timeout is triggered, the packet is locally retransmitted from the cache. To avoid unnecessary retransmissions during network problems the number of local retransmissions is limited to four. If the timeout is triggered by mistake (e.g. due to a bit error in the packet forwarded by the successor) and the TCP segment is retransmitted locally, the successor filters and drops this packet. End-to-end retransmissions are not to be filtered.
It is crucial for TSS that TCP ACKs are not lost since RTT estimation, retransmssions and caching depend on them. Therefore as with DTC TCP ACKs are regenerated locally and additionally an aggressive TCP acknowledgement recovery mechanisms is in place. For this mechanism to work each node measures the time between sending of an ACK and forwarding by the successor node. If after twice this average value the successor did not forward the ACK, it is recovered using the highest acknowledgement number stored in the TCP connection's state information.
TSS avoids packet forwarding in congestion situation and waits until a bit-error packet has been recovered. Hence a node does not forward more packets until it knows that the successor has received and forwarded all previously sent packets. Consequently, if this situation occurs not only does this node stop forwarding packets but all its predecessor nodes too. This is called the backpressure mechanism. The additional packet buffer avoids packets from being lost because the other buffer is full. When the successor of the head of the line node recovers the packet and forwards it, all other nodes will resume their usual modus operandi.

TSS example

Figure 5 shows the main principle of TSS' local retransmissions. While the first packet is received by node V without any problems (resulting in the ACK), a transmission error occurs with the second packet between node III and node IV. Node II, however, overheard node III forwarding the second packet and thusly forwards the third packet to node III. This is where the additional packet buffer comes into play: Node

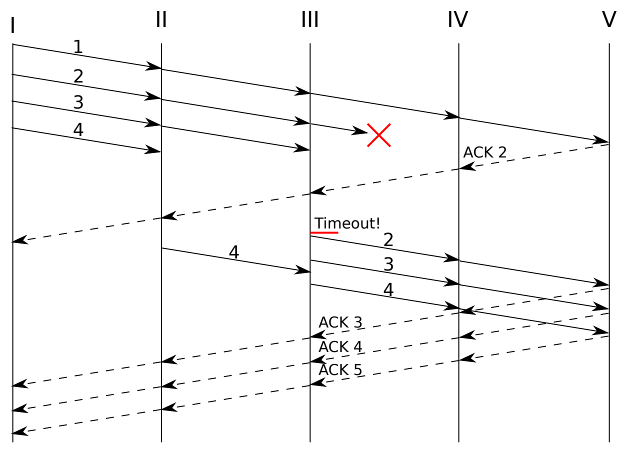

Figure 5: TSS example sending 4 packets over 5 nodes.

III stores the third packet received from node II in the additional buffer and does not forward any more packets. This leads to the so called backpressure mechanism: Node II does not forward any more packets itself, as it did not overhear node III's forwarding of the third packet. Only when the timeout at node III is triggered the packet forwarding continues. Node III retransmits the second packet to node IV, overhears node IV forwarding the same packet right away and continues with forwarding the third packet (which was stored in the additional packet buffer). Node II on the other hand overhears node III's forwarding of the third packet and forwards the fourth packet. From there on all packets are received by node V, the TCP endpoint, which then sends TCP ACKs for all received packets.

4.4 DTC and TSS performance

For high packet error rates (10 – 15 %) normal TCP performs extremly poor. DTC's and TSS' performance is similarly well, while DTC is doing slightly better with high error rates. For a packet error rate of 15 %, 500 TCP segments and 11 hops DTC transmitted about 14,500 TCP segments and ACKs and TSS 13,500 segments and ACKs. Generic TCP on the other hand used over 45,000 TCP transmissions. Assuming that transmission and reception of packets is the main source of energy consumption DTC as well as TSS can reduce that energy consumption of TCP/IP in wireless sensor networks by nearly 70 %.
Despite the differences between TSS and DTC (SACK vs. overhearing, 1 buffer vs. 2 buffers, etc.) both approaches perform similarly well. It should be noted that overhearing should not have a big impact on energy consumption as most packets are forwarded right away and the overhearing period is thusly very short [5].

4.5 Link Quality Estimated TCP

Link Quality Estimated TCP is another variation to the Snoop protocol [2] presented by Ponmagal et al. [16]. The Snoop protocol needs theoretically infinite buffer space at the gateway where all packets are cached. To address this issue Link Quality Estimated TCP employs a selective caching policy thus utilizing the gateway's buffer space more efficiently. TCP segments and ACKs flowing through the gateway to the TCP destination are cached only if the channel's condition is thought to be bad, i.e. when the probability of a transmission error is high. If this probability exceeds a certain threshold the packet is cached. The probability of a transmission error can be calculated as

$$P_{err} = 1 - (1 - P_1)F \cdot (1 - P_2)F$$

when transmitting a packet in two fragments with F being the fragment size and P_1 and P_2 being the expected bit-error rate for the first fragment and the second fragment respectively. It is left unexplained [16] why the two fragments would have different bit-error rates. Furthermore there is no indication on how to estimate the bit-error rate. This could be done through monitoring previous transmissions and averaging the loss rate. This mechanism, however, would be very slow in reacting to sudden changes in channel conditions. If a duplicate TCP ACK from the TCP destination is received at the gateway the corresponding packet (if cached) is retransmitted locally and all following packets are discarded. If the local retransmission is triggered by a timeout following packets are not discarded.
The rate-controlled wireless link selects its link rate according to some performance objective taking into account the current channel conditions. That means that the highest possible link rate which is below a specified bit-error rate threshold is selected.
Moreover the TCP window which is calculated as the minimum of the congestion window and the receiver window is modified. Along the way of ACKs returning from the receiver to the sender intermediate nodes adjust the receiver window to be as close to the bandwith-delay product as possible. This makes the TCP window adaptable to bottleneck situations in the network.
Ponmagal et al. [16] claim that their approach reduces TCP data transmission times by 5 % compared to normal TCP. Also end-to-end retransmissions are reduced from 18 to 4. This comes at a cost, however, as the gateway needs to store 14 packets whereas with normal TCP no storing is needed. Unfortunately the performance analysis setup (network topology, error rates, file sizes, number of runs,...) is not specified which makes it very hard to draw a consistent conclusion.

4.6 MSS Tuning

The link layer frame size limit in WSNs is relatively small (e.g. 127 bytes in IEEE 802.15.4 [11]). Therefore a TCP/IP packet may have to be fragmented before it can be transmitted in the network. Ayadi et al. are surveying the impact of fragmentation and other TCP parameters on energy consumption [1]. IETF's 6LoWPAN [14] introduces a new protocol layer between the IPv6 layer and the MAC layer. The 6LoWPAN layer compresses the IPv6 header and fragments the IPv6 packet to fit shorter MAC frames. The main source of energy consumption in a WSN is the transmission and reception of data. Thence it directly depends on the total number of bits sent by the whole network. The total number of bits sent depends on several factors: the number of hops between TCP source and TCP destination, the bit-error rate, the TCP maximum segment size (MSS), the maximum number of attempts on the link layer and forward error correction (FEC) redundancy ratio. FEC adds additional information to sent packets which helps recover the original packet if only a few bits are flipped. Therefore a retransmission can be avoided. However, adding FEC data adds an overhead to the packet.
When looking at one-hop transmissions there are three possible outcomes:

1. Failure: The data frame is lost. The sender will initiate a retransmission after a timeout.

2. Partial failure: The data frame is received correctly, but the ACK frame is lost. The sender will (uselessly) initiate a retransmission after a timeout.

3. Success: Both data and ACK frame are received correctly.

The expected total number of bits sent can be calculated analytically by applying probability theory to the different variables [1].
In multi-hop scenarios there are two potential outcomes:

1. End-to-end failure: After the maximum number of attempts on the link layer a node was still unable to forward a frame to the next hop.

2. End-to-end failure: The frame arrives at the TCP destination. Partial failures (i.e. less than the maximum number of retransmissions on the link layer) are possible.

In experiments Ayadi et al. compared the energy footprint of two different TCP maxium segment size choices: MSS = 64 bytes and MSS = 512 bytes. With the former no fragmentation is needed, whereas with the latter MSS the 6LoWPAN layer divides each segment into 8 frames. Using compression an IPv6 header can be shrunk to 2 bytes (from its initial 40 bytes) [13]. Their results showed that for a high bit-error rate it is recommended to use short TCP segments. This is due to the fact that when a segment is lost (due to the high error rate) all of its fragments need to be retransmitted from end to end. If the bit-error rate is rather low it is better to use larger MSS sizes. This leads to fewer acknowledgement messages sent which reduces energy consumption. If packets are fragmented due to a large MSS it is also advisable to increase the maximum number of link layer retransmissions. Thusly the number of costly end-to-end retransmissions can be reduced. On the other side, however, network or node problems are then not detected as quickly as with fewer retransmissions.
When looking at forward error correction they found that an optimal value for the FEC ratio exists [1]: Below this value adding redundancy reduces the loss probability leading to a lower energy consumption. Above this value the redundancy overhead is greater than the expected reduction in data loss. With a high FEC ratio packets with a large MSS perform better than with a small MSS (due to the reduction in transmission errors).
Finally their last finding is that with an increasing number of hops it is better to use small MSSs. This is due to the fact that the overall probability of a transmission error increases with a longer source to destination path.

5. CONCLUSION

In this paper an overview of the current status of TCP/IP in WSNs was given. Optimizing TCP/IP in a WSN is a very active research topic as TCP/IP faces many challenges before being fully adoptable in a sensor network. Various proposals exist for different scenarios. The most promising seem to be Distributed TCP Caching (DTC) [9] and TCP Support for Sensor networks (TSS) [4]. Those two approaches try to make TCP more energy efficient by tweaking the caching process, local retransmissions and introducing additional mechanisms. DTC as well as TSS can reduce the number of sent TCP segments by up to 70 % in respect to normal TCP. Fragmentation is also crucial in lossy networks and should be kept to a minimum when the error rate is high and the end-to-end path is rather long.
Since TCP/IP is the de-facto networking standard it is undoubtful that this protocol suite will be further adapted in such ways that small sensor nodes can cope with it. It is, however, unclear how long it will take to make the final push for TCP/IP adaption in WSNs.

6. REFERENCES

[1] A. Ayadi, P. Maillé, and D. Ros. TCP over low-power and lossy networks: tuning the segment size to minimize energy consumption. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1–5. IEEE.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 2–11. ACM, 1995.

[3] blip authors. blip — Berkeley IP implementation for low-power networks. `http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip`, 2011.

[4] T. Braun, T. Voigt, and A. Dunkels. TCP support for sensor networks. In *Wireless on Demand Network Systems and Services, 2007. WONS'07. Fourth Annual Conference on*, pages 162–169. IEEE.

[5] T. Braun, T. Voigt, and A. Dunkels. Energy-efficient TCP operation in wireless sensor networks. *PIK Journal Special Issue on Sensor Networks*, 28(2):93–100, 2005.

[6] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6), 1995.

[7] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98. ACM, 2003.

[8] A. Dunkels, J. Alonso, T. Voigt, H. Ritter, and J. Schiller. Connecting wireless sensornets with TCP/IP networks. *Wired/Wireless Internet Communications*, pages 583–594, 2004.

[9] A. Dunkels, T. Voigt, J. Alonso, and H. Ritter. Distributed TCP caching for wireless sensor networks. In *Proc. of the Mediterranean Ad Hoc Networking Workshop (MedHoc-Net)*. Citeseer, 2004.

[10] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM, 2003.

[11] IEEE Computer Society. 802.15.4 IEEE Standard for Information technology — Telecommunications and information exchange between systems — Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), 2003.

[12] M. Kuorilehto, J. Suhonen, M. Kohvakka, M. Hannikainen, and T. Hamalainen. Experimenting TCP/IP for low-power wireless sensor networks. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–6. IEEE, 2006.

[13] N. Kushalnagar, G. Montenegro, D. Culler, and J. Hui. Transmission of IPv6 Packets over IEEE 802.15. 4 Networks. 2007.

[14] N. Kushalnagar, G. Montenegro, C. Schumacher, et al. 6lowpan: Overview, assumptions, problem statement and goals. *draft-ietf-6lowpan-problem-01. txt, IETF Internet Draft (Work in progress)*, 2005.

[15] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options: RFC 2018. *Internet Request For Comments*, 1996.

[16] R. Ponmagal and V. Ramachandran. Link quality estimated TCP for wireless sensor networks. *International Journal of Recent Trends in Engineering*, 1(1):495–497, 2009.

[17] J. Postel. Internet protocol. RFC 791, IETF, September 1981.

[18] J. Postel. Transmission control protocol. RFC 793, IETF, September 1981.