Distributed TCP Caching for Wireless Sensor Networks

Adam Dunkels, Juan Alonso, Thiemo Voigt
Swedish Institute of Computer Science
{adam,alonso,thiemo}@sics.se

Hartmut Ritter
Freie Universität Berlin
hritter@inf.fu-berlin.de

Abstract— **Many applications of wireless sensor networks are useful only when connected to an external network. Previous research on transport layer protocols for sensor networks has focused on designing protocols specifically targeted for sensor networks. The deployment of TCP/IP in sensor networks would, however, enable direct connection between the sensor network and external TCP/IP networks. In this paper we focus on the performance of TCP in the context of wireless sensor networks.**

TCP is known to exhibit poor performance in wireless environments, both in terms of throughput and energy efficiency. To overcome these problems we introduce a mechanism called Distributed TCP Caching (DTC). The DTC mechanism uses segment caching and local retransmissions to avoid expensive end-to-end retransmissions. We show by simulation that DTC significantly improves TCP performance so that TCP can be useful even in wireless sensor networks.

I. Introduction

Wireless sensor networks consist of resource-constrained and sensor-equipped devices that communicate untethered. The networks are used for tasks such as monitoring and control. Many wireless sensor networks are not useful when operated in isolation; the sensor network must be connected to an external network through which monitoring and controlling entities can reach the sensors. The ubiquity of TCP/IP has made it the de-facto standard protocol suite for wired networking. By running TCP/IP in the sensor network it is possible to directly connect the sensor network with a wired network infrastructure, without proxies or middle-boxes [9]. It is often argued that the TCP/IP protocol stack is unsuited for sensor networks because of the specific requirements and the extreme communication conditions that sensor networks exhibit. We believe, however, that by using a number of optimizations, the performance and the energy consumption of TCP/IP can be greatly improved while at the same time benefiting from the ease of interoperability and generality of TCP/IP [8]. Furthermore, running TCP/IP in the sensor network also provides the possibility to route data to and from the sensor network using standard IP-based technologies such as General Packet Radio Service (GPRS) [4].

We envision that data transport in an *IP sensor network* is done using the two main transport protocols in the TCP/IP stack: the best-effort UDP and the reliable byte-stream TCP. Sensor data and other information that do not require reliable transmission is sent using UDP. TCP is used for administrative tasks that require reliability and compatibility with existing application protocols. Examples of such administrative tasks are configuration and monitoring of individual sensor nodes, downloads of binary code and data aggregation descriptions to sensor nodes.

This paper introduces the Distributed TCP Caching (DTC) mechanism which increases TCP performance in wireless sensor networks. TCP has previously been shown to have serious performance problems in wireless networks [3]. Moreover, the end-to-end acknowledgment and retransmission scheme employed by TCP causes expensive retransmissions along every hop of the path between the sender and the receiver if a packet is lost. DTC overcomes these problems by caching TCP segments inside the sensor networks and by local retransmission of TCP segments.

We show by simulation that DTC significantly improves TCP performance in wireless sensor networks in several ways:

- DTC substantially reduces the overall number of TCP segment transmissions.
- DTC decreases the number of end-to-end retransmissions.
- DTC shifts the burden of the energy consumption from nodes close to the base station into the network.

The last point is important since nodes close to the base station usually are the first to run out of energy [1], [13].

While we are not aware of any research on TCP/IP for wireless sensor networks, there is a plethora of work being done on TCP/IP for mobile ad-hoc networks (MANETs). There are, however, a number of differences between sensor networks and MANETs that affect the applicability of TCP/IP. MANET nodes are operated by human users, whereas sensor networks are intended to be autonomous. The user-centricity of MANETs makes throughput the primary performance metric, while the per-node throughput in sensor networks is inherently low because of the limited capabilities of the nodes. Instead, energy consumption is the primary concern in sensor networks. Finally, TCP throughput is reduced by mobility [11], but nodes in sensor networks are usually not as mobile as MANET nodes.

The rest of the paper is outlined as follows: In the next Section we discuss other issues that need to be tackled to make TCP/IP usable in wireless sensor networks. Section III gives an overview on Distributed TCP Caching while the following section discusses more details. In Section V we present simulation results. Further issues with DTC are discussed in Section VI. Before concluding the paper in Section VIII we present related work in Section VII.

II. Background

While improving TCP performance is necessary to make TCP usable in wireless sensor networks, there are other problems with TCP/IP that need to be tackled before TCP/IP can be deployed in such networks. We briefly describe the problems and our envisioned solutions in this chapter. We have previously discussed these in more detail [8].

IP addressing architecture. In ordinary IP networks, IP addresses are assigned to each network interface that is connected to the network. Address assignment is done either using manual configuration or a dynamic mechanism such as DHCP. In large scale sensor networks, manual configuration is not feasible and dynamic methods are usually expensive in terms of communication. Instead, we propose a *spatial IP address assignment* scheme that use spatial location information to provide semi-unique IP addresses to sensor nodes.

Header overhead. The protocols in the TCP/IP suite have very large headers, particularly compared to specialized sensor network communication protocols. The shared context nature of sensor networks makes *header compression* work well as a way to reduce the TCP/IP header overhead.

Address centric routing. Routing in IP networks is based on the addresses of the hosts and networks. Due to the application specific nature of sensor networks, data-centric routing [10] is often preferable to address-centric routing. We use a specific form of *application overlay networks* to implement data-centric routing and data aggregation for TCP/IP sensor networks.

Limited nodes. Sensor nodes are typically limited in terms of memory and processing power. It is often assumed that the TCP/IP stack is too heavy-weight to be feasible for such small systems. In previous work [7], we have shown that this is not the case. Our μIP implementation of the TCP/IP stack [6] can be run 8-bit micro-controllers and requires only a few hundred bytes of RAM.

III. Overview on Distributed TCP Caching

The reliable byte-stream TCP was designed for wired networks where bit-errors are uncommon and where congestion is the predominant source of packet drops. Therefore, TCP always interprets packet drops as a sign of congestion and reduces its sending rate in response to a dropped packet. Packet drops in wireless networks are often due to bit-errors, which leads TCP to misinterpret the packet loss as congestion. TCP will then lower the sending rate, even though the network is not congested.

Furthermore, TCP uses end-to-end retransmissions, which in a multi-hop sensor network requires a retransmitted packet to be forwarded by every sensor node on the path from the sender to the receiver. As Wan et al. note, end-to-end recovery is not a good candidate for reliable transport protocols in sensor networks where packet loss rates are in the range of 5% to 10% or even higher [21]. A scheme with local retransmissions is more appropriate since it is able to move the point of retransmission closer towards the final recipient of the packet.

To deal with these issues, we propose a scheme called *Distributed TCP Caching* (DTC) that is based on segment caching and local retransmissions in co-operation with the link layer. Other mechanisms for improving TCP performance over wireless links, such as TCP Snoop [3], focus on improving TCP *throughput*. In contrast, DTC is primarily intended to reduce the *energy consumption* required by TCP. DTC does not require any protocol changes neither at the sender nor at the receiver. Rather, DTC resides only in the intermediate nodes in the sensor network.

We assume that each sensor node is able to cache only a small number of TCP segments; specifically, we

assume that nodes only have enough memory to cache a single segment.

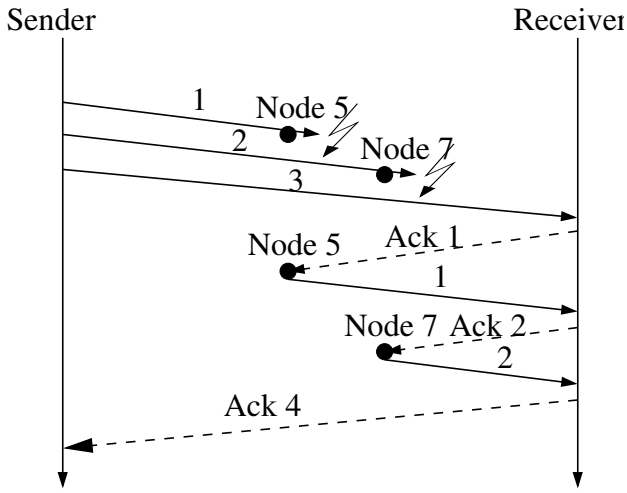

Fig. 1. Distributed TCP Caching

Figure 1 shows a simplified example of DTC. To keep the example simple, we assume that nodes are able to detect when a TCP segment they have transmitted is lost. The algorithms for packet loss detection are described in the next Section.

In this example, a TCP sender transmits three TCP segments. Segment `1` is cached by node 5 right before it is dropped in the network, and segment `2` is cached by node 7 before being dropped. When receiving segment `3`, the TCP receiver sends an acknowledgment (`ACK 1`). We assume here that node 7 must not retransmit segment `2` when it receives `ACK 1` since this acknowledgment comes too early. Retransmitting segments too fast can lead to spurious retransmissions as explained in the next Section.

When receiving `ACK 1`, node 5, which has a cached copy of segment `1`, performs a local retransmission. Node 5 also refrains from forwarding the acknowledgment towards the TCP sender, so that the acknowledgment segment does not have to travel all the way through the network. When receiving the retransmitted segment `1`, the TCP receiver acknowledges this segment by transmitting `ACK 2`. On reception of `ACK 2`, Node 7 performs a local retransmission of segment `2`, which was previously cached. This way, the TCP receiver obtains the two dropped segments by local retransmissions from sensor nodes in the network, without requiring retransmissions from the TCP sender. When the acknowledgment `ACK 4` is forwarded towards the TCP sender, sensor nodes on the way can clear their caches and are thus ready to cache new TCP segments.

IV. DTC Implementation

In this Section we present our current DTC design and in particular the algorithms for caching and retransmission of segments.

A. Segment Caching and Packet Loss Detection

DTC uses segment caching to achieve local retransmissions. Because of the memory limitations of the sensor nodes, it is vital to the performance of the mechanism to find an appropriate way for nodes to select which segments to cache. A desirable outcome of the selection algorithm is that segments are cached at nodes as close to the receiver as possible, and that nodes closer to the receiver cache segments with lower sequence numbers. To achieve this, each node caches the TCP segment with the highest sequence number seen, and takes extra care to cache segments that are likely to be dropped further along the path towards the receiver. We use feedback from a link layer that uses positive acknowledgments to infer packet drops on the next-hop. Our design works with either active or passive acknowledgments [12]. A TCP segment that is forwarded but for which no link layer acknowledgment is 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 an acknowledgment that acknowledges the cached segment is received, or when the segment times out.

To avoid end-to-end retransmissions, DTC needs to respond faster to packet drops than regular TCP. DTC uses ordinary TCP mechanisms to detect packet loss: timeouts and duplicate acknowledgments. 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. The delays experienced by the nodes are lower than those estimated by the TCP end-points, and the nodes are therefore able to use lower timeout values and perform retransmissions earlier than the connection end-points.

In TCP, duplicate acknowledgments signal either packet loss or packet reordering. A TCP sender uses a threshold of three duplicate acknowledgments as a signal of packet loss, which may be too conservative for DTC. Since each DTC node inspects the TCP

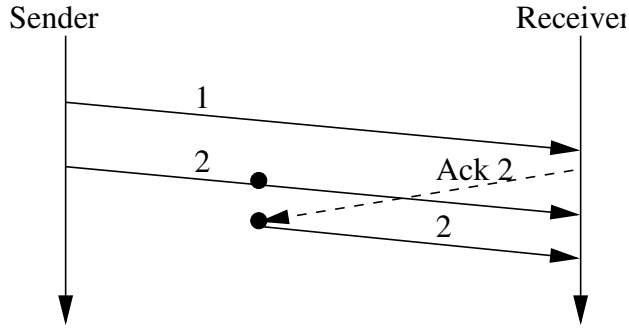

Fig. 2. Spurious retransmission

sequence numbers of forwarded TCP segments, the nodes can compute a heuristic for the amount of packet reordering. If the nodes see that packet reordering is uncommon in the network, they can lower the duplicate acknowledgment threshold. Furthermore, DTC tries to avoid spurious retransmissions caused by misinterpreting acknowledgments for new data as acknowledgments that signal packet loss, as shown in the Figure 2. The nodes use estimated round-trip times to distinguish between an acknowledgment that detects a lost packet and one that acknowledges new data.

DTC uses the TCP SACK option to both detect packet loss and as a signaling mechanism between DTC nodes. DTC uses the latter to inform other nodes about the segments in their cache.

B. Algorithm outline

The algorithm we are currently using for DTC nodes is the following:

1.**on** receiving data packet with seq. number `seqNr`:
  if cache not locked
    cache with certain probability
  if cache is empty
    cache
  forward unless node has retransmitted same `seqNr` within some time limit
  set link level ACK timeout

2. **on** link level ACK timeout:
  lock cache
  set retransmission timer

3. **on** receiving ACK with acknowledgment number `ackSeq` and sequence number `cached` in cache:
  update local RTT
3.1 if `ackSeq` > `cached`
    cancel retransmission timer
    clear cache
3.2 if `ackSeq` = `cached`
    if time since last transmission > local RTT ∗ factor
      retransmit cached segment
3.3 if `ackSeq` < `cached` and SACK option set
    clear cache if `cached` in SACK block
    if `cached` not in SACK block
      retransmit
      add cached seqNr to SACK block
      if all gaps filled
        drop ACK
3.4 if `ackSeq` < `cached` and no SACK option set and cache locked
    add SACK option with cached seqNr as SACK block

4. **on** local retransmission timeout:
  retransmitted cached segment
  set retransmission timer

When an intermediate node receives a segment and the cache is not locked, the node caches the segment only with a probability of 50%. Our simulations have shown that, as expected, this leads to a better distribution of cached segments than caching every new segment when the cache is not locked.

DTC interprets link level timeouts as a strong indication that the data segment has been lost. When a link level acknowledgment timeout occurs, DTC locks the cache and sets a retransmission timer for local retransmission of the cached segment.

Action 3.2 is sensitive to packet reordering, as it may erroneously interpret packet reordering as packet loss. Because DTC nodes inspect all TCP segments, it is possible for them to infer if packet reordering occurs. Action 3.2 may then only be triggered if no packet reordering has been seen. However, our simulations have shown that this action is not critical for the performance of DTC: it reduces the number of transmitted messages with less than 1% when no packet reordering occurs. This means that in a real-world implementation we will not implement this action.

Figure 3 shows the example in Figure 1 using SACK. When receiving segment `3`, the receiver sends an acknowledgment (`ACK 1`) with a SACK block for segment 3. When receiving this acknowledgment, node 7 retransmits segment `2`. According to action 3.3, node

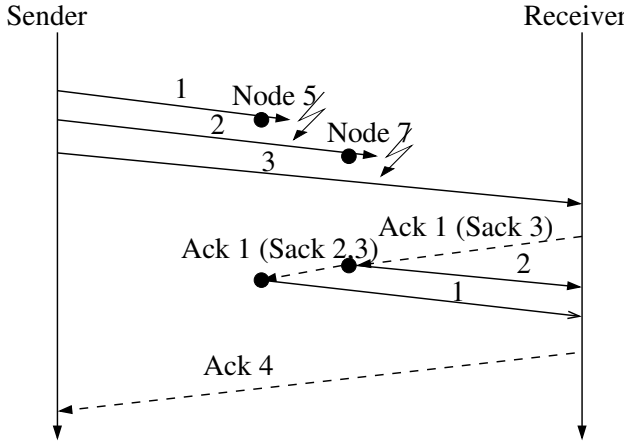

Fig. 3. DTC with SACK

7 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 does not forward the acknowledgment according to the last condition in action 3.3.

The two actions 3.3 and 3.4 also illustrate how DTC uses SACK as a signaling mechanism. A selective acknowledgment option indicates either that the receiver has received an out-of-order segment, or that a DTC node closer to the receiver has locked the segment in its cache. A DTC node that sees a selective acknowledgment for a segment it has in its cache can therefor clear the cache.

Note that DTC does not require that the sender actually uses SACK. The first DTC node, seeing a SYN segment without the SACK option could set the SACK option. This node would also remove the SACK options of all segments traveling in the opposite direction. Since DTC nodes already keep TCP state for example for local round trip times and sequence numbers, this only requires little extra state.

C. Flying Start

Sensor nodes cannot handle large amounts of data because of their constrained resources. Therefore, even "bulk" data transfers in sensor networks are quite short. Since DTC uses round-trip time measurements, it needs a good estimation of the round trip time as quickly as possible. Towards this end, we have implemented a scheme called *flying start*. Flying start makes the TCP SYN segments hop-by-hop reliable and each node measure the round-trip time during the TCP connection set-up. This round-trip time measurement is used as an initial estimation of the TCP round-trip time. Our simulations show that flying start increases the efficiency of DTC with about 10-25% compared to using the default initial values of TCP's retransmission timer. TCP's initial values are by nature quite conservative in order to avoid unnecessary retransmissions. The initial round-trip times produced by the flying start mechanism also naturally takes the current network conditions into account. Packet losses lead to a higher estimated round trip time because of the hop-by-hop retransmissions. Our simulations demonstrate that the initial round trip time estimations using flying start are quite accurate.

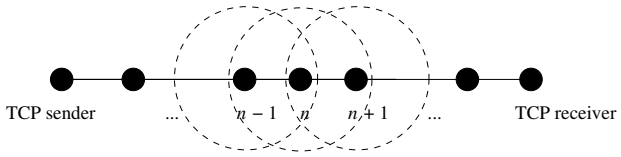

Fig. 4. Simulation topology

V. Results

We have implemented DTC and performed evaluations in the OMNet++ discrete event simulator [20].

We have performed simulations with unidirectional TCP data transfers, with and without the DTC mechanism enabled. The data transfers consisted of 500 TCP segments. As a comparison, 500 segments with 100 bytes payload correspond to roughly 50 kilobytes, which is slightly less than the flash memory of the ESB sensor nodes [2]. We use a chain topology as shown in Figure 4 where node n is in transmission range of node $n-1$ and $n+1$, but node $n-1$ is not in range of node $n+1$. We ran the simulations until the sender receives the acknowledgment for the 500th segment. For our simulations, we have implemented a link layer with explicit positive link layer acknowledgments.

Since TCP data segments are larger than acknowledgments, we set the packet loss probability for data segments to twice the loss probability for TCP acknowledgments and to four times the loss probability of link level acknowledgments. We use a uniformly distributed packet loss model.

Our simulations consist of 30 runs, and the reported results are the average of the 30 runs. The results indicate that DTC brings vast improvements: For path lengths between 6 and 11 hops and per-hop packet loss rates between 5% and 15%[1], the number of end-to-end

[1]When not explicitly noted, we mean per-hop packet loss when we say packet loss.

retransmissions performed by the sender decreases by a factor of ten. The amount of end-to-end retransmissions decrease even more for higher packet loss rates and longer paths.

A. Transmitted Messages: With and Without DTC

Wireless communication is often the major power consumer during sensor operation [15]. Therefore, the number of packet transmissions is one indicator of the energy efficiency of a network protocol given a MAC layer that schedules packets accordingly.

Tables I and II present the total number of data segment and acknowledgment transmissions for a network of 6 and 11 hops. The total number of transmitted segments counts all transmissions of packets in the network, not only the initial transmissions from the sender. For example, a packet traveling over 6 hops counts as 6 transmissions. The tables show that the number of data packet transmissions is much lower with DTC than without. For 6 hops and low packet loss rates DTC reduces the number of data segment transmissions with about 15%, while for 11 hops and high packet loss rates the reduction is about two thirds.

In particular, the number of end-to-end retransmissions decreases with a factor of almost 30. This shows that DTC is very effective in avoiding end-to-end retransmissions.

DTC also reduces the number of transmitted acknowledgments. It is interesting to note that the number of transmitted acknowledgments is lower for packet loss rates of 5% and 10% than when there is no packet loss. The reasons for this are that TCP's cumulative acknowledgment strategy is robust against loss of acknowledgment and that DTC reduces the number of acknowledgment transmissions. In fact, an acknowledgment lost close to the receiver reduces the overall energy consumption if the acknowledgment for the next segment in the same window does not get lost.

It should be noted that we assume that the TCP receiver does not use delayed acknowledgments. While delayed acknowledgments could reduce the number of acknowledgments, they also impact the nodes' estimated local round trip times negatively.

Our results show that with DTC much fewer packets are transmitted for the transport of the same amount of data.

B. Transmitted Messages: Comparison with Theoretical Bounds

While our simulation results show that DTC significantly increases TCP performance, we cannot quantitatively evaluate how large the increase is. TCP performance in wireless networks is known to be poor, and we are not aware of comparable work on TCP performance in wireless sensor networks.

Instead, to be able to produce a reasonable comparison we have developed analytical models of two idealized reliable transmission protocols. These models allow us to compute theoretical lower bounds on the achievable performance and enable us to evaluate the performance of DTC.

We have developed analytical models for the following two idealized protocols:

- an end-to-end scheme with local retransmissions and *perfect knowledge* about packet losses, and
- a reliable hop-by-hop scheme.

a) End-to-end scheme with local retransmissions and perfect knowledge about losses: We first consider an *omniscient* protocol that is able to provide a reliable channel by local segment caching and *perfect knowledge* about packet losses. The protocol would have the right segment cached and retransmit it from the right node when a packet is lost. The protocol would transmit exactly one extra packet for each lost data packet. Such a protocol might not be possible to implement in a real system, but is useful as a reference since it provides a lower bound on the achievable performance.

In our setup, this protocol would require $500n/(1-p)$ segment transmissions, where p is the per-hop loss probability for data segments and n is the number of hops. 500 is the number of segments to be transfered. For example, with a packet loss rate of 10%, the protocol must make at least 3333 transmissions for 6 hops and 6111 for 11 hops. As shown in Table I and Table II, DTC needs 3755 transmissions (6 hops) and 7795 transmissions (11 hops). For 6 hops DTC is only 12% above this theoretical lower bound and about 27% for 11 hops.

b) Reliable hop-by-hop scheme: Consider a reliable hop-by-hop scheme with a loss rate of p for data packets and q for acknowledgments. The number of required data packet transmissions by each hop for one packet is given by

$$1 + ((1-p)q + p) + ((1-p)q + p)^2 + \\ + ((1-p)q + p)^3 + ... = \\ = \frac{1}{1 - ((1-p)q + p)}.$$

The term $(1-p)q+p$ is the probability that the segment must be retransmitted, because either the segment is lost (with probability p), or the segment is correctly received

Packet loss rate	0%	5%	10%	15 %
Transmitted data packets without DTC	3010	3819	5142	7307
Transmitted data packets with DTC	3010	3285	3755	4736
DTC transmitted ACKs	3003	2884	2954	3370
TCP without DTC end-to-end retransmissions	0	159.1	446	1030.6
DTC end-to-end retransmissions	0	55.6	108.6	173.8

TABLE I
COMPARISON DTC VS.NON-DTC WITH 6 HOPS

Packet loss rate	0%	5%	10%	15 %
Transmitted data packets without DTC	5515	8788	16109	33012
Transmitted data packets with DTC	5515	5844	7795	11228
DTC transmitted ACKs	5508	4835	5596	7140
TCP without DTC end-to-end retransmissions	0	388.4	1579.6	5088
DTC end-to-end retransmissions	0	66.8	135	208.8

TABLE II
COMPARISON DTC VS. NON-DTC WITH 11 HOPS

but the acknowledgment is lost (with probability $(1-p)q$).

To be able to compute the number of acknowledgments that must be transmitted we set $x = (1-p)q + p$. The number of required acknowledgment transmissions is then given by

$$(1-p) + (1-p)x + (1-p)x^2 + (1-p)x^3 + ... = \\ = \frac{p}{1-x}.$$

Since the loss probability of a data segment is p, data segments are correctly received with probability $1-p$. Whenever a data segment is received an acknowledgment is transmitted. With probability x the data segment must be retransmitted and if so, we must also transmit another acknowledgment. The probability of this is $(1-p)x$.

Over a path with 11 hops, a data segment loss rate of 10%, an acknowledgment loss rate of 5%, and with 500 segments to be transported the hop-by-hop scheme must transmit 6433 data segments and 5789 acknowledgments. In the same situation, DTC transmits 7795 data segments and 5596 acknowledgments (see Table II). Thus, DTC requires about 21% more data segment transmissions but about 4% less acknowledgment transmissions than the hop-by-hop scheme over 11 hops.

When transmitting data over 6 hops, the hop-by-hop scheme transmits 3508 data segments and 3158 acknowledgments. Comparing this with DTC (see Table I), we see that the number of data segment transmissions is less than 10% above the hop-by-hop number, but the number of acknowledgment transmissions is slightly lower. The examples show that with respect to acknowledgments DTC's end-to-end scheme is in some scenarios better than the hop-by-hop scheme. The major reason for this is that TCP's cumulative acknowledgment strategy tolerates some acknowledgment loss.

TCP is based on positive acknowledgments. Therefore, we have not compared DTC with a hop-by-hop scheme that uses negative acknowledgments such as PSFQ [21].

C. Load Reduction Near Sender

In sensor networks, sensor data flows from nodes that collect sensor data to sinks, whereas control or management data flows from sinks to sensor nodes [21]. Therefore, nodes close to the sink usually are the first to run out of energy because sensor data has to be routed through them them [1], [13]. Thus, a transport protocol should shift the burden from these nodes to nodes inside the network. Performing local retransmissions instead of end-to-end retransmissions could obviously assist in that task.

In our simulations we have counted the number of transmissions of data segments each node has to perform. Figure 5 shows the results for 11 hops and a packet loss rate of 10% for data packets. In the figure, the numbers on the y-axis denote the average number of transmissions per run and node. In the figure, node 0 is the node closest to the TCP sender (sensor data sink), node 9 the node closest to the TCP receiver. The figure

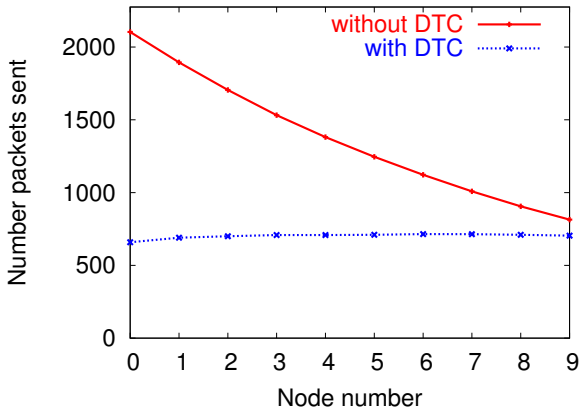

Fig. 5. DTC load reduction near sender

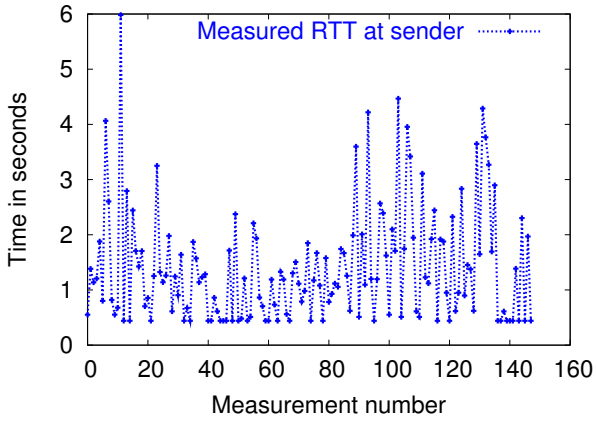

Fig. 6. Measured RTT by TCP sender, with DTC

shows that without DTC, nodes close to the sink have to transmit much more segments than nodes further away from the sink. With DTC, load is reduced at nodes close to the sink and evenly distributed among the nodes on the path. In fact, using DTC, the vulnerable nodes close to the sink perform slightly *less* transmissions than nodes close to the receiver.

D. Round Trip Times

Since DTC uses local retransmissions, it affects TCP's round-trip time (RTT) estimations. When a lost segments is retransmitted inside the network, the round trip times measured by the sender will increase and vary more than without DTC. A varying RTT and thus a varying retransmission timeout (RTO) can cause two main problems. First, the RTO may be too high which leads to lower throughput. We will later show that despite the varying round trip times, DTC actually increases the throughput. Second, if the RTT is too low, unnecessary end-to-end retransmissions may occur. An unnecessary end-to-end retransmission is a retransmission made by the TCP sender for a segment that is present in the network but has not reached the TCP receiver. We can identify unnecessary end-to-end retransmissions in our simulations by make two simulation runs and explicitly set a much higher RTO at the TCP sender in one of the simulation runs. By comparing the number of retransmissions made in the two simulations we can see how many unnecessary retransmissions were done. By performing such experiments, we have seen that unnecessary end-to-end retransmissions rarely occur.

Figure 6 shows the measured RTT with DTC during a typical run with 11 hops and 10% data packet loss rate. While the figure shows the exact RTT, most TCP

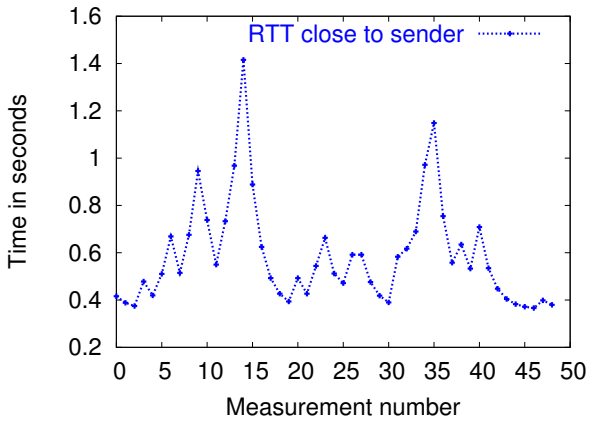

Fig. 7. Local RTT close to sender

implementations do not measure these values as fine-grained. Without DTC, the TCP sender always measures a short RTT (the minimum time in Figure 6 which is slightly below 0.5 seconds). As expected, the figure shows that with DTC the measured RTT varies a lot. The variation is due to DTC's local retransmissions, which cause the measured RTT to vary depending on how often and where a segment is retransmitted within the network.

Figure 7 and Figure 8 show the local RTT measured by one DTC node close to the sender (Figure 7) and one DTC node close to the receiver (Figure 8). The local RTT is defined as the RTT between the DTC node and the TCP receiver. As expected, the local RTT measured close to the receiver is lower and varies to a lesser extent.

E. Throughput

In wireless sensor networks with low communication bandwidth, resource-constrained nodes, and high packet loss rates, TCP throughput cannot be expected to be

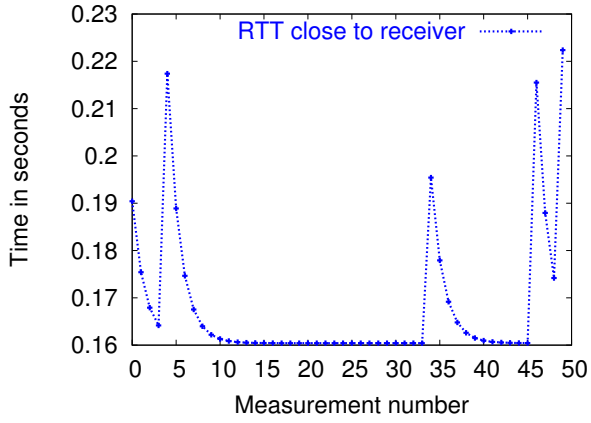

Fig. 8. Local RTT close to receiver

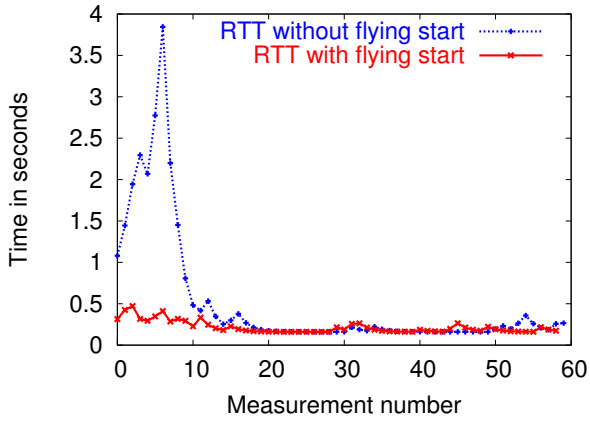

Fig. 9. Local RTT with and without flying start

high. For the administrative tasks we consider as TCP's primary use in sensor networks, throughput is however of secondary importance. Still, we expect that DTC, by performing local retransmissions, can increase the TCP throughput.

Number of hops	6	11
Throughput improvement	450%	670%

TABLE III
THROUGHPUT IMPROVEMENT WITH DTC

Table III shows TCP throughput improvement for 6 and 11 hops with 10% packet loss rate. The results show that DTC indeed achieves much higher throughput than without DTC.

F. Flying Start

This simulation experiment captures the effect of the flying start scheme described in Section IV.

Figure 9 shows the local RTT measured at a DTC node close to the receiver, with and without flying start. The figure clearly demonstrates the advantage of using flying start: the node's RTT reaches a stable estimate much faster with flying start.

It seems reasonable that quickly reaching a good RTT estimation should improve throughput, in particular for short data transfers. We have measured the time it takes to transport 20 packets with and without flying start over 6 and 11 hops with a data segment loss rate of 10%. Table IV shows that flying start decreases the transfer time substantially.

Number of hops	6	11
Duration decrease (%)	32.3	39

TABLE IV
DTC SHORT BULK TRANSFER DURATION DECREASE

VI. DISCUSSION

In this Section we discuss further issues with TCP and DTC in sensor networks.

A. Applicability of Unicast in Sensor Networks

There are many examples where a reliable unicast transport protocol is useful in wireless sensor networks. One example is the need for reprogramming certain groups of sensors, e.g. in a geographical area. In that case one could unicast the new binary to a designated node in that area which would then broadcast it to the regional subnet.

Another example is that nodes might need a new task list once they are done with part of their task. Using TCP a sensor node could download a new task list and the associated code from an external TCP/IP network. Since not all nodes or clusters might have fulfilled their tasks at the same time unicast communication is appropriate.

B. Memory Consumption

Our scheme assumes that sensor nodes have three buffers: a receive and transmit buffer as well as a cache. Because we assume that the TCP receiver is part of the sensor network, it can set an appropriate maximum segment size (MSS) for the TCP connection. This ensures that TCP segments do not exceed the size of these buffers. The receiver can impact the maximum number of segments in flight by choosing MSS as well as the size of the offered window. On reception of an

out-of-order segment, the receiver must be able to buffer segments, for example in an EEPROM.

C. Packet Loss Rates and Routing

DTC is designed to handle packet loss but performs better when packet loss rates are low. Woo et al. have developed a routing protocol that is able to find stable routes with low loss rates [22]. They achieve end-to-end success rates larger than 80% over 6 hops in an office environment. The authors find that in uncongested networks the routes are fairly stable. These results indicate that the packet loss rates we have used in our simulations are realistic even for indoor office environments that are potentially harsh for wireless communication [23].

While choosing a routing scheme that favors stable routes seems advantageous for TCP, we have not yet quantified the impact of route changes. However, after a routing update, the caches of new nodes on the path are empty, and therefore, DTC would initially behave like TCP but not worse. To improve the situation after route changes, we could apply hop-by-hop reliability similar to flying start.

VII. Related Work

DTC can be seen as a generalization of the Snoop Protocol [3]. Snoop provides local retransmissions on the last hop from a base station to a mobile entity. DTC extends this idea towards multi-hop sensor networks.

There are two basic categories for reliable transport protocol for wireless sensor networks. The first is to transport sensor readings in a reliable way from sources to the sink(s), the second is to transport data from the sink(s) to one, several or all sensor nodes.

Stann and Heidemann's RMST belongs to the first category [18]. RMST is a reliable data transport layer protocol for sensor networks. Like DTC, this protocol can be configured for in-network caching. RMST is specifically designed to run on top of directed diffusion. The ESRT transport protocol aims at reliable event detection with minimum energy expenditure [16]. In ESRT, the sink monitors the event-to-sink reliability and adapts the reporting periodicity of the sources accordingly. Unlike DTC, RMST and ESRT are designed for data collection.

There are a few approaches for transport layers that transport data from the sink to the sources. Wan et al. have developed PSFQ [21] while Park and Sivakumar have proposed another solution that delivers entire messages with reduced time-delay compared to PSFQ [14]. While the protocols are designed to achieve high efficiency, DTC aims at providing interoperability with external TCP/IP networks. This makes it possible to directly connect the sensor network with a wired network infrastructure, without proxies or middle-boxes [9].

Donckers et al. have designed an energy-efficient transport protocol suitable for a wireless link between base station and mobile host in a split-connection scheme [5]. While they separate the wired and wireless connection to gain energy efficiency, our aim is to avoid a connection split.

Several others have examined the energy consumption of different TCP variants, both analytically [24], via simulation [19] and experimentally in a wireless testbed [17].

VIII. Conclusions and Future Work

In this paper, we have presented Distributed TCP Caching. DTC enhances TCP performance in sensor networks both in terms of energy efficiency and throughput. DTC achieves this by caching TCP segments inside the sensor network and retransmitting lost segments locally. Furthermore, DTC shifts the burden of the load from vulnerable nodes close to the base station into the sensor network. There are more ideas and trade-offs to be explored. For example, we have not yet studied the potential gains that might be achieved by using a more reliable link layer. We also need to investigate how DTC behaves in the context of multiple TCP flows.

We are currently implementing the DTC mechanism on actual sensor nodes in order to measure real-world performance and preliminary results show that the sensor nodes are capable of running both a full TCP/IP stack and the DTC mechanism.

IX. Acknowledgments

This work was partly financed by VINNOVA, The Swedish Agency for Innovation Systems.

References

[1] J. Alonso, A. Dunkels, and T. Voigt. Bounds on the energy consumption of routings in wireless sensor networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Cambridge, UK, March 2004.

[2] CST Group at FU Berlin. Scatterweb Embedded Sensor Board. Web page. Visited 2004-02-21. `http://www.scatterweb.com/`

[3] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP performance over wireless networks. In *MOBICOM'95*, Berkeley, California, November 1995.

[4] J. Cai and D. Goodman. General packet radio service in GSM. *IEEE Communications Magazine*, 35:122–131, October 1997.

[5] L. Donckers, P. Havinga, G. Smit, and L. Smit. Energy efficient TCP. In *Proceedings 2nd Asian International Mobile Computing Conference (AMOC2002)*, Malaysia, May 2002.

[6] A. Dunkels. The uIP TCP/IP Stack for Embedded Microcontrollers. Web page. Visited 2004-05-21. `http://www.sics.se/~adam/uip/`

[7] A. Dunkels. Full TCP/IP for 8-bit architectures. In *MOBISYS'03*, San Francisco, California, May 2003.

[8] A. Dunkels, T. Voigt, and J. Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Work-in-Progress Session of the first European Workshop on Wireless Sensor Networks (EWSN 2004)*, Berlin, Germany, January 2004.

[9] A. Dunkels, T. Voigt, J. Alonso, H. Ritter, and J. Schiller. Connecting Wireless Sensornets with TCP/IP Networks. In *WWIC2004*, February 2004.

[10] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.

[11] G. Holland and N. Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *MOBICOM '99*, August 1999.

[12] J. Jubin and J. Tornow. The darpa packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[13] V. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff. Design of surveillance sensor grids with a lifetime constraint. In *First European Workshop on Wireless Sensor Networks (EWSN 2004)*, Berlin, Germany, January 2004.

[14] S.-J. Park and R. Sivakumar. Sink-to-sensors reliability in sensor networks. In *Poster presentation at MobiHoc 2003*, Annapolis, MD, USA, June 2003.

[15] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002.

[16] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT : Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proceedings of the 4th ACM international symposium on mobile ad hoc networking and computing (MobiHOC 2003)*, 2003.

[17] H. Singh and S. Singh. Energy consumption of TCP reno, newreno, and SACK in multi-hop networks. In *ACM SIGMETRICS 2002*, 2002.

[18] F. Stann and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, Anchorage, Alaska, USA, April 2003.

[19] V. Tsaoussidis, H. Badr, X. Ge, and K. Pentikousis. Energy / throughput tradeoffs of TCP error control strategies. In *5th IEEE Symposium on Computers and Communications (ISCC)*, 2000.

[20] A. Varga. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference*, Prague, Czech Republic, June 2001.

[21] C.Y. Wan, A. T. Campbell, and L. Krishnamurthy. PSFQ: A Reliable Transport Protocol For Wireless Sensor Networks. In *WSNA'02*, Atlanta, September 2002.

[22] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *The First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, California, November 2003.

[23] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *The First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, California, November 2003.

[24] M. Zorzi and R. Rao. Is TCP energy efficient? In *IEEE International Workshop on Mobile Multimedia Communications*, San Diego, California, 1999.