An integrated approach to developing sensor network solutions *

Adam Dunkels Laura Marie Feeney Björn Grönvall
Thiemo Voigt
Computer and Network Architecture Laboratory
Swedish Institute of Computer Science
Box 1263, SE 164 29 Kista, Sweden
{adam,lmfeeney,bg,thiemo}@sics.se

Abstract

This paper describes a prototype sensor networking platform and its associated development environment. Key elements of the system are the ESB sensor hardware, the Contiki operating system, and the communication stack, which includes a MAC layer and a highly optimized TCP/IP. Because the work is driven by prototype applications being developed by project partners, particular attention is paid to the development environment and to practical deployment issues. Three prototype applications are also presented.

1 Introduction

In this paper, we describe the ongoing development of sensor network solutions at the Computer and Network Architecture Laboratory of the Swedish Institute of Computer Science. Because the work is driven by application experiments that are being developed by project partners, it is necessary to take an integrated approach to the development of a fully functional system. Another result of this approach is that development environment and deployment issues become a "first-class" component of our work. Because of the variety of available sensor devices, portability is also a key concern.

In the following sections, we present four main elements of our solution and describe current applications. The core of the system is the ESB sensor hardware running the Contiki operating system. A Contiki emulation/simulation environment has also been developed to aid in the development process.

The communication stack includes an adaptive, energy efficient MAC protocol and a TCP/IP layer that has been optimized for resource constrained devices. Providing support for TCP/IP allows the sensor network to be connected to the Internet infrastructure and enables key management functionality such as debugging and upgrading sensor applications.

Although some elements, in particular the MAC layer and some TCP enhancements, are still being explored in simulation, the current prototype is sufficiently functional to support simple experimental deployments. Three such experiments are outlined in the final section: a building security monitoring; marine environmental measurements; and energy conservation in a residential complex.

2 Hardware platform

We use the ESB sensor board [4], developed by the Freie Universität Berlin as a hardware platform for prototyping and development (Figure 1). The ESB board consists of a Texas Instruments MSP430 low-power micro-controller, an RF Monolithics TR1001 single-chip RF transceiver, and a collection of sensors. The MSP430 has 60 kilobytes of flash ROM and 2 kilobytes of RAM. A 32 kilobyte serial EEPROM provides additional persistent secondary storage. The RF transceiver operates at 868 MHz and supports transmission rates up to 115.2 kbps. The board is equipped with two external connectors: an RS-232 port and a JTAG interface. The RS-232 port enables communication with external devices such as

Figure 1: Hardware platform: ESB from FU Berlin

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

a PC, while the JTAG interface can be used for code downloading and debugging. The on-board sensors are:

- A light sensor for the detection of visible light.
- A passive infrared sensor for detection of movement.
- A temperature sensor.
- A vibration sensor for detection of movement of the sensor board.
- A microphone for determination of the ambient noise level.
- An infra-red sender and receiver.

The MSP430 micro-controller is designed specifically for low-power applications and provides a set of low-power sleep modes. The micro-controller is awakened from the sleep mode by an interrupt, generated either by one of the internal timers, or by an external device such as one of the sensors.

To enable run-time reprogramming, the MSP430 supports selective rewriting of the internal flash ROM. This feature, combined with mechanisms in our Contiki operating system, drastically shortens the development cycle.

The TR1001 RF transceiver implements baseband transmission with either amplitude shift keying or on-off keying. The transceiver module provides half-duplex bit level access to the physical radio medium. All higher level mechanisms, such as MAC protocol processing, data encoding, and time multiplexing, must be done in software. However, because the transceiver is connected to one of the MSP430 UARTs, bit-shifting is done by the hardware rather than in software. This reduces the load on the micro-controller, as the UART causes an interrupt only after a full 8-bit byte has been received. Other sensor node designs, such as the Berkeley MICA motes [1], do not have their RF transceiver connected to a UART, and therefore an interrupt in generated for each incoming bit.

3 The Contiki Operating System

Operating systems for small embedded devices are commonly designed to offer high predictability and therefore require that resources and processes are predetermined and pre-allocated at compile time. In such systems, the operating system and all applications running on top of the system are statically linked into a monolithic binary that is downloaded into the embedded device. This approach works well for traditional embedded systems, but does not provide the flexibility needed for distributed sensor networks. Fixed, pre-determined resource allocation and static linking make it hard to dynamically update sensor node software, both during development and in deployed systems.

In contrast, the Contiki [2] operating system is designed with flexibility in mind and allows individual programs and services to be dynamically loaded and unloaded in a running system.

Like other operating systems for sensor devices, such as TinyOS [9], Contiki is based on an event-based concurrency model. Unlike other systems, however, Contiki also provides preemptive multi-threading for applications that specifically require it.

Event-based systems have the advantage of lower resource requirements than thread-based systems. Also, they are well suited to sensor networks due to the event-driven nature of the networks themselves. There are, however, applications that do not work well in event-based systems. One class of such applications are those that involve cryptographic computations. On the slow microprocessors used in sensor networks, such computations may take many seconds to complete. In a purely event-based system, the system is not able to respond to external events during the computation. In Contiki, the computation can be performed in a separate thread, allowing events to be handled while the computation runs in the background.

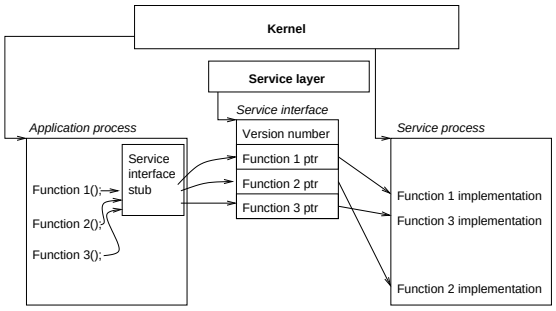

Figure 2: Dynamic linking of a service

When developing applications for sensor networks, the ability to reprogram the sensors without requiring physical access to the nodes greatly simplifies development and reduces the development time. Contiki has support for loading individual programs from the network, which makes it possible to dynamically reprogram the behavior of the network. A thin service layer, conceptually situated next to the kernel (Fig-

ure 2), provides service discovery and run-time dynamic service replacement within each sensor node.

Contiki is designed to be portable across a wide range of different platforms. We have ported Contiki to several platforms, including the Atmel AVR and the Intel x86. Contiki was ported to the Z80 platform in less than a day, by a third-party developer.

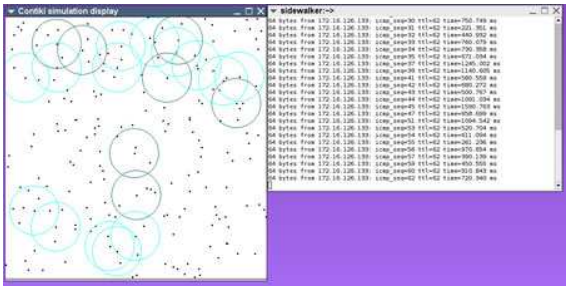

Figure 3: Screenshot of the Contiki simulator

The inherent portability of Contiki also makes it trivial to run Contiki as a user-level process under a PC operating system such as FreeBSD. We have implemented a Contiki simulation environment in which each sensor node is represented by its own FreeBSD process and connected by network layer simulation. Because the simulator runs the same Contiki code as the ESB hardware, application programs developed in the simulator can be directly compiled and run on the sensor hardware. A screenshot of the simulator is shown in Figure 3.

4 MAC layer

Because it operates between the network layer and the wireless transceiver and among nodes sharing a wireless channel, the MAC layer plays a key role in two important areas: energy efficiency and quality of service. We are developing a lightweight MAC [8] in which an energy efficient TDMA-like structure is overlaid on a CSMA-based collision avoidance protocol. Distributed adaptation to traffic and interference can provide support for simple non-signalled quality of service.

This approach has several features that make it highly appropriate for sensor networks. Most importantly, the proposed MAC is asynchronous[7][11]. Avoiding synchronization overhead is particularly important for resource limited sensor nodes, which must also meet strict requirements on size, complexity and cost. Moreover, sensor networks are often intended for deployment in extreme or highly variable environments, which can negatively affect hardware stability.

The proposed MAC is also lightweight. In the absence of higher layer traffic, the MAC generates no traffic overhead – an advantage that is enabled by forgoing synchronization. This property is particularly important for sensor networks that are intended to monitor an environment over an extended period.

Because the proposed MAC uses adaptive mechanisms to minimize contention and interference, it is able to provide good best-effort QoS, as well as simple non-signalled (better than best-effort) QoS. Because no centralized coordination is used, it is also scalable for multihop sensor networks.

The simple underlying mechanism is extensible and additional mechanisms can be overlaid on it. In future work, it should be possible to support more complex signalled QoS schemes, on-demand synchronization, and admission control.

4.1 Energy efficiency

In sensor nodes, the radio transceiver consumes a significant proportion of system energy. Although transmitting presents the highest energy consumption, energy consumed in receiving and idle (awake and prepared to receive) modes cannot be treated as negligible. The latter may be particularly significant, because nodes spend only a small proportion of their time transmitting and receiving.

A centralized network, in which the central node has knowledge of cell traffic and (often) access to electrical mains, naturally supports mechanisms based on spending energy and computational resources at this node in order to save them in the more constrained remote nodes. In particular, the central node can schedule transmissions to minimize the amount of time that each node must spend awake and waiting for traffic, as well as acting as a source of (internal) synchronization for the network.

In a decentralized network that relies on cooperative forwarding of sensor data to a gateway, this approach is less directly applicable. Because nodes cannot predict when they will receive data for forwarding, they must remain in the high energy consumption idle mode. Moreover, there is no natural source of synchronization for traffic announcements or neighbor discovery messages, which can be used to establish a virtual infrastructure, e.g. [3].

An asynchronous power save protocol is based on the following observation: If each node remains awake just over half $(.5 + \epsilon)$ of the time, its awake interval will overlap with that of each of its neighbors. In addition, the overlap is guaranteed to include either the first ϵ or last ϵ of the awake interval – regardless

of their phase difference [1].

Any node with pending data uses these two intervals to broadcast a traffic indication (ATIM) message, as well as to transmit any other broadcast traffic. The ATIM message indicates the neighbors for which the transmitter has pending traffic, along with its current estimate (if any) of that neighbor's relative phase. By the principle outlined above, every neighbor receives one or the other of the ATIMs.

If the phase estimate in the ATIM is wrong, due to either phase change or clock drift, or is missing, the destination provides this information with an ATIM-ACK message. From the phase information, the sender can determine its available "transmission window" (Figure 4) for each neighbor. Note that the ATIM mechanism is only used to determine intervals during which two neighbors are both awake to communicate. Access to the channel itself can be based on any CSMA mechanism; current work is based on the DCF defined in IEEE 802.11.

The energy saving obtained by this mechanism is roughly $(.5-\epsilon)$, less the overhead. However, it is possible to overlay two (or more) instances of this mechanism, using integral multiples of the base period. The increased latency and number of retransmissions required to ensure phase discovery make this approach most suitable for networks with low data frequency compared to the network lifetime.

4.2 Flow adaptation

A multihop wireless sensor network is subject to a number of forms of contention and interference which significantly reduce its effective capacity[10]. The mechanism described above only allows neighbors to determine their transmission windows; it does not ensure that all pending data can be transferred during a transmission window.

Clearly, the phase distribution among a group of nodes significantly affects the available capacity of a link or region. For example, Figure 4 shows a felicitous distribution of phases. Because the transfer windows of the three nodes do not overlap, they cannot contend with each other. Moreover, because the transmission window A-B is followed by the transmission window B-C, a flow A-B-C will experience low latency.

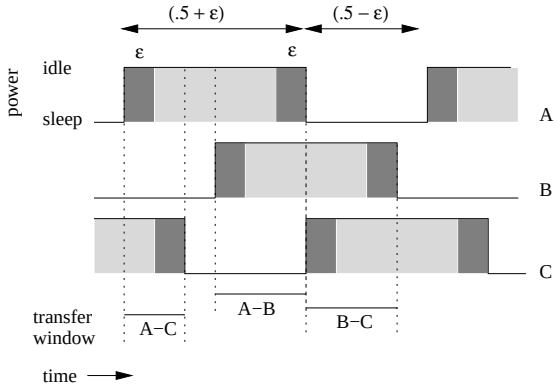

Figure 4: Transfer windows. *ATIM* (dark) and transfer interval (light) are shaded.

One advantage of the phase discovery approach is that a node can seamlessly change its phase with respect to its neighbors. Phase adjustment can be used to increase the effective capacity of a region and reduce latency along a path.

For example, a node can adjust its phase in order to avoid attempting to send data when there are high levels of contention or interference. Alternatively, the sequence of nodes which forms a path can adjust their phase in order to minimize intra-path interference and minimize latency.

The MAC protocol is currently being evaluated in simulation, using random phase adjustment. This approach has the advantage of simplicity, as well as reducing the likelihood of problematic feedback interactions. Future work includes more complex schemes, as well as incorporating some form of admission control.

5 Network and Application Layer: TCP/IP for Sensor Networks

In addition to application-specific distributed sensing and data fusion algorithms and protocols, there is still a requirement for supporting TCP/IP in the sensor network environment. This requirement is driven by practical issues such as network management, calibration, diagnostics, and debugging. Moreover, running standard Internet protocols in a sensor network makes it possible to connect the network directly to the Internet.

[1]Formally: Define a period $I = 1$ and and a value $0 < \epsilon \leq 0.25$. Let T and R independently schedule an awake interval of duration $(0.5 + \epsilon)$ followed by a sleep interval of duration $(0.5 - \epsilon)$, with a phase difference of $0 \leq \phi < 1$. Either the sub-interval $[0..\epsilon]$ or the sub-interval $[0.5..0.5+\epsilon]$, measured on T, is completely contained in the awake interval of R.

The proof is trivial: Measured on T, the awake interval of T is $[0..0.5+\epsilon]$ and the awake interval of R is $[0+\phi..0.5+\epsilon+\phi]$. If $0 \leq \phi < 0.5$, the awake interval of R cannot begin after $t = 0.5$ and cannot end before $t = 0.5 + \epsilon$. If $0.5 \leq \phi < 1$, the awake interval of R cannot begin after $t = 1$ and cannot end before $t = 1 + \epsilon$.

Because a practical wireless sensor network cannot be operated in isolation, it must somehow be connected to an external network so that monitoring and management entities can communicate with it. There are significant practical advantages to being able to connect a sensor network to external IP-based infrastructures without the need for special proxies or middle-boxes [6]. We are therefore working on adapting the standard TCP/IP protocol stack for the specific needs of wireless sensor networks.

We envision that the UDP and TCP protocols are used for all data transport in sensor networks. Sensor data is generally transmitted using UDP/IP in conjunction with application specific data aggregation protocols. Certain administrative tasks require reliable unicast connections, as well as the ability to address a particular node. For this purpose, TCP/IP is used.

One example of such an administrative function is upgrading or reprogramming an individual sensor. For example, it may be necessary to provide a new task list to a node once it has completed its current task. Because different nodes may finish their tasks at different times, or only a subset of nodes are to be upgraded, it may be necessary to address individual sensors. Using TCP, a new task list and the associated code can be sent to a sensor from an external IP network. This technique reduces the amount of code that needs to be stored on the device, as well as providing additional flexibility.

Another important example of administrative functionality for which TCP/IP is useful are debugging and diagnostic tasks requiring reliable connectivity to a specific sensor. The importance of this kind of supporting functionality quickly becomes clear when deploying experimental applications and fully justifies the effort spent on this case.

It is sometimes argued that TCP/IP is too heavy weight for resource constrained sensor networks. Among the issues which raise concern are the memory overhead, the header overhead, the appropriateness of the IP addressing scheme, as well as the overall performance impact of the TCP end-to-end acknowledgment and retransmission scheme. We address the first two issues below and present the last two in more detail in sections below.

We have developed uIP [5], a complete TCP/IP implementation with a code size of only a few kilobytes and requiring only few hundred bytes of RAM. Note that a focus on a specific implementation does not preclude portability: uIP runs not only on the ESB sensor platform, but has been ported to a variety of 8-bit and 16-bit processors.

Issues of header overhead can be overcome by using specific forms of header compression that utilize the fact that the data transmitted in the sensor network will be generated by a small set of applications.

5.1 Spatial IP addressing

In sensor networks, it is generally required that nodes have a notion of their location, in order to give the collected data meaning. In spatial IP addressing, each sensor node uses its spatial location to construct its IP address, as shown in Figure 5. It is important to note that the spatial IP address not necessarily denotes a single identifiable sensor node. Rather, the address only represents the location of the node. If a node is replaced, the new node will assign itself the same IP address as the replaced node. Nodes that move are simply assigned a new IP address based on their current location.

Location information may be an absolute position (i.e. from a GPS unit) or it may be a relative position derived from some localization protocol. The address may even be derived from a logical position, such as a room number, as in the building security application described below.

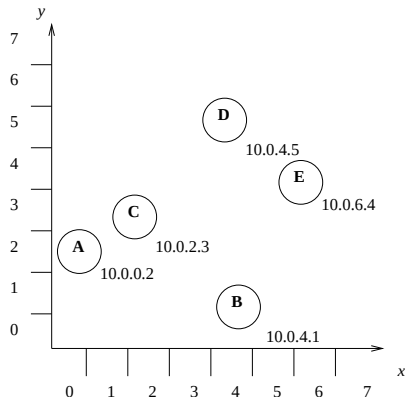

Figure 5: Spatial IP address assignment

If nodes are aware of their spatial location and use this information to derive their address, then address assignment requires neither a central server nor communication among the sensor nodes. Furthermore, spatial IP addressing enables regional subnetting, as well as various forms of regional broadcast, as in Figure 6.

5.2 Distributed TCP Caching

The TCP end-to-end acknowledgment and retransmission scheme results in expensive retransmissions along every hop of the path between the sender and

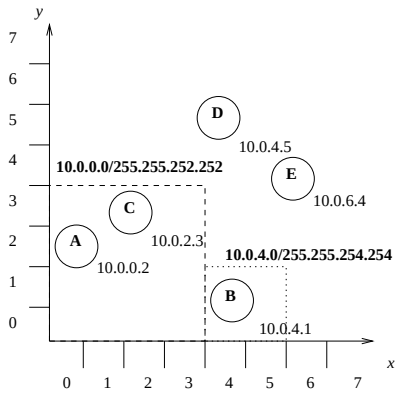

Figure 6: Regional subnets

the receiver if a packet is lost. This leads to poor performance in terms of both energy consumption and throughput. To overcome these problems, we have developed *Distributed TCP Caching (DTC)*.

Distributed TCP caching avoids end-to-end retransmissions by caching TCP segments in the network and performing local retransmissions.

Ideally, each node would cache all the segments and the last node to have successfully received a segment would perform a local retransmission. Because sensor nodes are memory constrained, however, each node is only permitted to cache one segment. Nodes therefore attempt to identify and cache segments that may not have been received successfully by the next hop, as indicated by a missing (active or passive) acknowledgment.

In our simulations, we use active link layer acknowledgments. A TCP segment that is forwarded but for which no link layer acknowledgment is received is assumed to 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 when an acknowledgment referring to the cached segment is received, or when the segment times out.

To avoid end-to-end retransmissions, DTC needs to respond to lost packets more quickly than regular TCP. DTC uses ordinary TCP mechanisms to detect packet loss: timeouts, duplicate and selective acknowledgments. Every node participating in DTC therefore has to maintain a soft TCP state for each end-to-end connection for which it forwards packets.

Simulation experiments show that this mechanism substantially improves TCP performance in wireless sensor networks: DTC significantly reduces the overall number of TCP segment transmissions and the number of end-to-end retransmissions. Analytical results also suggest that DTC performs well: The number of transmitted segments is only about 10% higher than would occur given an ideal caching strategy for short paths (six hops) and about 25% higher for longer ones (11 hops).

A high-level network simulation environment allows developers to quickly and easily generate large numbers of randomized scenarios. Having studied the effectiveness of DTC in this environment, we are currently in the implementing it for the ESB nodes, using the Contiki simulator.

6 Applications

This section describes three prototype networks being developed in cooperation with project partners in their areas of specific interest and presents some preliminary results. Although the communication patterns within the networks themselves are fairly simple, a number of interesting issues arise in practical deployments.

6.1 Building security

The purpose of this sensor network application is to secure a sensitive installation. The work is carried out in cooperation with SAAB Technologies. A demonstration was held in May, 2004.

In the demonstration scenario, a network of motion sensors is deployed in an office environment. If there are movements in the building that are not supposed to be there, they are detected by the sensor network and an alarm is sent via GSM. A security team responding to the alarm logs into the building network to obtain information about the status inside the building.

The demonstration network was deployed along a 90 meter long corridor, and about ten adjacent offices. The network consisted of ESB nodes with two separate roles: motion detector nodes and backbone nodes. The motion detectors were placed in the offices and the backbone nodes were placed along the corridor. Each motion detector had a direct communication path to at least one backbone node, and the backbone nodes each had contact with one other backbone node. One backbone node was equipped with an external network interface. Overall, the demonstration network was relatively small, consisting of two to four backbone nodes and five to eight sensors.

When a motion detection node detects movement, it transmits an alarm message to the closest backbone node. The backbone node saves this message

and transmits the information to its neighbor backbone nodes. Eventually, all of the backbone nodes have information about the entire state of the network. The backbone node with network connectivity transmits the alarm information over GSM.

The security team also has a mobile backbone node (a PDA) that can contact the closest backbone node, download the entire network state, and provide a "tactical display".

The network used a variant of spatial IP addressing where all nodes, both motion detection nodes and backbone nodes, were given (x, y) coordinates and configured themselves with an IP address in the form 172.16.x.y. This configuration information was provided to each sensor at deployment time over IR using a backbone sensor as a form of remote control – a convenient solution to a practical problem in deploying sensor network solutions. The mobile backbone node had a fixed IP address from another network, indicating that it somehow was different from the other backbone nodes.

6.2 Marine monitoring

This work is intended to study the feasibility of using a sensor network for monitoring water temperature and salinity, significantly reducing the cost of this aspect of marine research. The application is currently being developed in cooperation with UmeåMarine Sciences Centre.

The network will be deployed in the northern part of the Baltic Sea, where fresh water movements affect temperature and salinity. The depth of the water is about 20 meters at the location being used for the study.

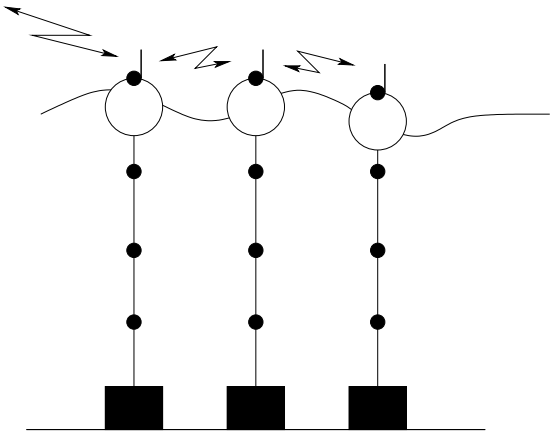

Figure 7: Marine sensor network

The structure of the network is shown in Figure 7. Sensors are attached to a cable that descends from a buoy, providing measurements of temperature and salinity at known depths. Because underwater communication using acoustic channels is still in its infancy and underwater modems are expensive, these sensors will be connected as a fixed network. The sensors being developed for underwater sensing are based on a simplified ESB node, waterproofed for use in a marine environment.

Above the waterline, at the buoy atop each cable, is a full function ESB node. These nodes collect sensor data from the fixed network beneath them and forward the data over their wireless interface to the node designated as the gateway. A GPRS module is attached to the gateway node via the RS-232 interface of the ESB.

The collected data will be transferred to the marine sciences center via GPRS. The GPRS interface also supports data transport to the sensor nodes for reprogramming, monitoring and debugging. This functionality is an example of the usefulness of being able to manage nodes directly via TCP/IP protocols.

6.3 HVAC Monitoring

The purpose of this network is to explore the feasibility of instrumenting a residential complex to improve the efficiency of its HVAC (heating, ventilating, and air conditioning) system. The network is being deployed by a small company, Raditex AB, which intends to use the information from this study to improve the energy efficiency of the heating system.

In order to know where the energy is consumed, it is necessary to measure the temperature at many different locations inside the buildings. This is done using a scaled down ESB node, equipped with less sensors than a full scale node: temperature and vibration.

The energy control rooms for the complex are already connected over Ethernet, so an IP-based sensor network fits well with the existing infrastructure. This network also uses a variant of spatial IP addressing, where the IP address includes the building number, the floor, and the number of the closest apartment.

7 Conclusion

In this paper, we have described the ongoing development of sensor network solutions at the Computer and Network Architecture Laboratory of the Swedish Institute of Computer Science. Key elements of the system are the ESB hardware platform, the Contiki operating system and the communication stack,

which includes a specialized TCP/IP that is suitable for resource constrained sensor networks.

The work has been driven by a variety of prototype applications, also described in the paper. The result is an emphasis on development environment and network management issues. Some emerging patterns are: the importance of an effective development environment for programming and debugging sensor systems, the versatility of the ESB sensor hardware and Contiki operating system as a prototype development platform, and the importance of paying sufficient attention to deployment, configuration and system maintenance issues.

All of the system software and development tools are freely available in source form (`http://www.sics.se/cna`) and the system has begun to attract a growing user community.

8 Acknowledgments

The authors would like to acknowledge the support of Jochen Schiller and his colleagues at the Freie Universität Berlin and the partners in the DTN/SN project.

References

[1] Berkeley mica motes. Web page. URL: http://www.xbow.com/ Visited 2004-06-22.

[2] Adam Dunkels and Björn Grönvall and Thiemo Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *First IEEE Workshop on Embedded Networked Sensors* (2004).

[3] Chen, B., Jamieson, K., Balakrishnan, H., and Morris, R. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *ACM Wireless Networks Journal 8*, 5 (Sept. 2002), 481–494.

[4] CST Group at FU Berlin. Scatterweb Embedded Sensor Board. Web page. URL: http://www.scatterweb.com/ Visited 2004-06-22.

[5] Dunkels, A. Full TCP/IP for 8 Bit Architectures. In *Proceedings of the First International Conference on Mobile Systems, Applications and Services (MobiSys)* (San Francisco, May 2003).

[6] Dunkels, A., Voigt, T., Alonso, J., Ritter, H., and Schiller, J. Connecting Wireless Sensornets with TCP/IP Networks. In *Proceedings of the Second International Conference on Wired/Wireless Internet Communications (WWIC2004)* (Frankfurt (Oder), Germany, Feb. 2004).

[7] Feeney, L. M. An asynchronous power save protocol for wireless ad hoc networks. Tech. Rep. T2002:09, SICS – Swedish Institute of Computer Science, July 2002. revised version February, 2003.

[8] Feeney, L. M. A QoS aware power save protocol for wireless ad hoc networks. In *Proceedings of the First Mediterranean Workshop on Ad Hoc Networks(Med-Hoc Net 2002)* (Sardenga, Italy, Sept. 2002).

[9] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems* (Nov. 2000).

[10] Li, J., Blake, C., Couto, D. S. J. D., Lee, H. I., and Morris, R. Capacity of ad hoc wireless networks. In *Proc. of 7th Annual International Conference on Mobile Computing and Networking* (2001), pp. 61–69.

[11] Tseng, Y.-C., Hsu, C.-S., and Hsieh, T.-Y. Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks. *Comput. Networks 43*, 3 (2003), 317–337.