Architecture Strategies for Energy-Efficient Packet Forwarding in Wireless Sensor Networks

Vlasios Tsiatsis, Scott A. Zimbeck, Mani B. Srivastava
Networked & Embedded Systems Laboratory, E.E. Dept, UCLA
Los Angeles, CA, 90095
{tsiatsis, szimbeck, mbs}@ee.ucla.edu

ABSTRACT
The energy-efficient communication among wireless sensor nodes determines the lifetime of a sensor network and exhibits patterns highly dependable on the sensor application and networking software. This software is responsible for processing the sensor data and disseminating the data to other nodes or a central repository. In this paper we propose a node architecture that takes advantage of both the intelligence of the radio hardware and the needs of applications to efficiently handle the packet forwarding. It exploits principles widely used in modern firewall network architectures and as our analysis shows achieves considerable energy savings.

Keywords
Energy-efficient packet forwarding, sensor networks

1. INTRODUCTION
Recently wireless ad-hoc sensor networks have gained considerable attention by the research community because of the new challenges they pose to researchers. The limited power and computational resources as well as the distinct types of data they carry and the data centric applications [2] running on them call for a different approach in constructing the overall sensor node architecture.

Figure 1 shows a simplified version of the overall sensor architecture widely adopted by researchers [6][9][11]. The major parts of a wireless microsensor system are: a) the sensor node processing subsystem running on the sensor node main CPU, b) the sensor subsystem, and c) the communication subsystem. The applications executing on a sensor node utilize all the subsystems to collect, process and receive (transmit) data from (to) other sensor nodes in the vicinity. Our proposed architecture strategies are based on the fact that the distinct communication layers shown in Figure 1 are not actually implemented in one board or device. The network communication functionality is split between the main CPU and the radio board, which are connected together by a slow serial packet link. We show later in the paper that it is because of this fact that a considerable amount of energy is wasted when packets cross the boundary between the two physical components. A forwarded packet crosses the slow serial link twice regardless of what part (application or network layer) determines the need for forwarding (Figure 1).

ISLPED '01, August 6-7, 2001, Huntington Beach, California, USA.

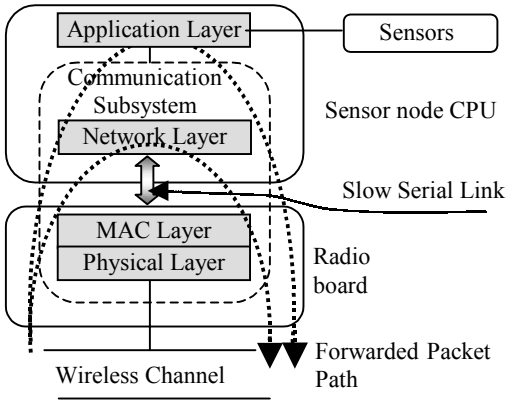

Figure 1. Typical Wireless Sensor Node Architecture

Although current radio boards have processing power in the form of a microcontroller unit (MCU) [6][9][11], this is only used for the physical and MAC layer implementation. We advocate that part of the functionality of the network layer can migrate from the main sensor CPU to the radio board in an application-defined manner. Devices that incorporate an MCU and some amount of Configurable Logic (FPGA) as the Triscend E520 [10] or the Atmel FPSLIC [5], can also be the host of a limited number of network layer functions. The advantage of these devices is that several operations such as link layer destination checking, CRC checking, can be performed in the configurable logic, thus alleviating the MCU and the main node CPU. Higher level and more complex packet processing (network layer specific) can also be performed in the MCU thus allowing more sophisticated packet filtering to take place nearer to the radio hardware. With the advent of these new technology devices, protocol specific processing can be performed in two stages, namely hardware and software inside the same device.

A significant number of packets are processed in a simple manner and forwarded to the next hop. For the sake of overall delay and power consumption these packets should be processed as near to the radio hardware as possible [1]. To show this we measured the percentage of received packets (both routing and data from all the nodes) that were accepted, dropped or forwarded for a specific scenario simulated on the NS-2 network simulator. The scenario consists of a sensor terrain of 1000x1000 units inside of which 30 sensor nodes are uniformly placed. The transmission range of the radio of each node is 250 units. Two nodes are picked at random to be the source and the sink of a session of a constant-rate flow

of packets. We used IEEE 802.11 MAC, and DSR [3] as the routing protocol. From this simulation we found that 65.567% of the packets were forwarded and, 34.3% of the packets were accepted. The rest were duplicate packets that were dropped.

Sensor networks are application-specific data dissemination networks. Individual applications should have the flexibility of defining their own methods of processing and routing their data and packets. In this paper we present a network communication subsystem architecture which employs multiple levels of packet processing in order to provide: a) Ability for the communication subsystem to stop or redirect packet flow in a sensor node as low in the protocol stack as possible and b) Methods for applications to define their own routing protocols at run time.

The rest of the paper is structured as follows: Section 2 describes the proposed architecture. Section 3 presents the analysis and measurements for our prototype. Finally Section 4 concludes the paper.

2. ARCHITECTURE

Our proposed application-defined forwarding architecture has its roots in the various packet filtering architectures like the BSD Packet Filter [4]. The user specifies the packet filter as a set of packet field matching rules connected together as an expression tree with AND/OR operators. The packet field matching rules designate the field of the packet to be checked (start byte offset, length, mask), the matching value and the matching operator.

Our packet processing architecture consists of two parts: the first located on the radio board and the second on the sensor node (Figure 2). We assume that the radio board contains a MCU as the main processing component and some amount of configurable logic (FPGA). The task of the radio board part is to drop or redirect received packets that, according to the rules dictated by the applications, should not enter the sensor node. The task of the sensor node part (Router Manager) is to perform more sophisticated packet processing and packet flow demultiplexing. Each application dictates the routing rules for each part by using a filter specification explained later in the paper. In this specification it designates the assignment of the rules to each part. We assume that the application programmer has knowledge of the sensor node resources and capabilities, and as a result, s/he should assign the necessary routing rules to the appropriate parts.

2.1 Rules and Actions

Applications specify their routing protocols by defining two components: the rules against which a desired packet is matched and the action(s) taken upon a packet match. Simple rules and simple actions are specified using tuples of the general form: {byte_offset, bit_length, bit_mask, value, operator}. Rules contain either comparison or ALU operations. Using the specified tuple fields, rules perform the operation, Packet[byte_offset : bit_length] & bit_mask OPERATOR value. All rules evaluate to either true or false. Comparison rules are tuples that specify operators that test for equality, inequality or bit settings. These operators are EQ, GT, GTE, LT, LTE, and SET which perform their respective tests on packets. ALU rules use supplied tuple information to perform the following arithmetic operations on packets: ADD, SUB, MUL, DIV, AND, OR, LSH, RSH, NEG. These rules always return true. The result of the evaluation is held in an accumulator A or a register X for evaluation using the above-mentioned comparison rules. Combining ALU rules allows filters to perform arbitrarily complex operations on packet data. Additionally, there are two operators, LD and ST that perform loads and stores to and from a packet offset, the accumulator, or the register.

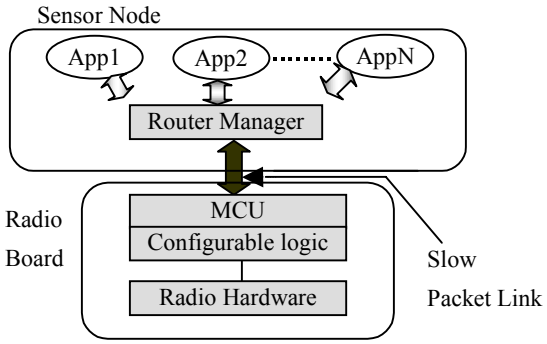

Figure 2. Two tier architecture

Actions can be of the following: terminating, and non-terminating. Terminating Actions are actions that, when encountered, stop packet filtering for the current packet. These actions are ACCEPT, DROP, and FORWARD_EX. ACCEPT is used when an application wishes to be the exclusive recipient of a matched packet. DROP simply drops the packet. FORWARD_EX (exclusive) results in the retransmission of the current packet over the radio channel. Non-terminating actions perform operations on a packet and allow subsequent rules and actions to be performed on the packet. These include modification actions, COPY, FORWARD_SH (shared), and RETURN. Modification actions allow the content of any packet to be changed. Modification is realized by using the ALU rules described above in conjunction with the LD and ST operators. COPY is similar to ACCEPT, except that the packet is allowed to continue in processing. In the same manner, FORWARD_SH allows the packet to be further filtered after being forwarded. RETURN is an action that merely marks the end of a sequence of rules and actions. Composite rules and actions are sequences of simple rules and simple actions joined by the logical operators AND/OR. An application will typically specify a number of rules and actions represented as tuples and a separate AND/OR expression tree whose leaf nodes are these specified tuples (see Figure 3). Packet processing, for a given packet and filter specification, is the evaluation of leaf-node rules and actions (tuples) on the data contained in the packet during the traversal of the application filter's expression tree. In Figure 3 we define two rule tuples namely A, B, which are connected together with AND/OR operators (*, + respectively) to construct the following composite rules: a) If rule A is false then the filter falls down to the Drop rule which drops the packet, b) If rules A and B are true then Action1 is executed.

2.2 Two-tier Packet Processing Architecture

Our two-tier architecture is presented in Figure 4. We assume that the radio hardware is a simple receiver/transmitter that is able to identify the start of packet and notify the configurable logic (CL) that this event has occurred. It is also capable of streaming the packet bits into the CL. Otherwise all the necessary functionality for packet framing within a continuous bitstream is implemented in the CL. Moreover, upon a packet transmission the CL should be able to notify the radio hardware that a packet is ready for

transmission and next be able to stream the packet bits into the radio hardware.

The block diagram of hardware packet processing component residing in the CL is depicted in Figure 4.

```
tuple_t Tuples[] = {
  {A_offset, A_length, A_mask, A_value, A_operator}, // A
  {B_offset, B_length, B_mask, B_value, B_operator}, // B
  {0, 0, 0, Action1_value, OP_Action1}, // Action1
  {0, 0, 0, 0, OP_DROP} // Drop
};
filter_tree_t App_Example_filter {
  "A*(B*Action1)+Drop;" //expression
};
```

Figure 3: Example of a C-code application-defined routing filter specification

The hardware architecture consists of the control unit (CU), several packet filed matchers (PFM), a receive (RX_FIFO) and a transmit FIFO (TX_FIFO), a CRC module and a Boolean vector. The CU is responsible for all the control signaling between the MCU and all the hardware modules residing in the configurable logic. This includes the programming of the PFMs and any packet transmissions from the TX_ FIFO. PFMs, are used to match incoming packet fields against fixed values or other packet fields. All the parameters for field matching (start bit, length, mask, fixed value, operator) are downloaded to the PFMs by the CU at the programming stage for the Configurable Logic. The CRC check module performs Cyclic Redundancy Check on every received packet and compares it with the corresponding value stored in the packet. The Boolean vector stores the outcome of the comparison operation performed in each PFM. The supported comparison operators are equal, not equal, greater than, less than, greater than or equal, less than or equal.

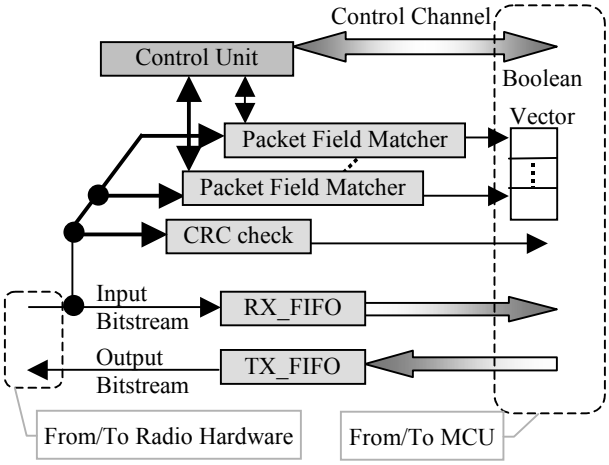

Figure 4. Configurable Logic Architecture

When signaled by the radio hardware that a new packet is being streamed in the CL, the control unit enables all the PFMs to start scanning the bitstream and perform the field matching according to their programmed values. At the same time the packet is being stored in the RX_FIFO queue and the CRC is being calculated. Shortly after the packet reception is finished, the control unit signals the MCU that a new packet is ready in the RX_FIFO queue. When signaled, the MCU extracts the packet from the FIFO and the boolean vector only if the CRC indicates a correct packet reception. One of the novelties of our architecture is the speed up of the rule matching part by the use of the boolean vector.

After receiving a signal from the hardware that a packet is ready, the MCU applies higher-level software packet filtering on the received packet. The MCU evaluates the expression tree produced from the filter specification. Some of the simple packet field matching rules are assigned to hardware (CL), and therefore, the outcomes of those rules already exist in the boolean vector. This contributes to the minimization of the delay for processing a packet and the power consumed by the MCU.

Clearly the MCU, being more flexible than the configurable logic, can perform more sophisticated operations on packets. Assisted by the CL, the MCU is capable of deciding if a packet is destined for the current node or if it must be forwarded, even for the cases that require complex calculations (e.g. distance calculation). Also the routing protocols that maintain little or no state on the sensor node are implemented in the MCU. Examples of these protocols are flooding without duplicate suppression.

The second tier of our architecture lies in the sensor node's central processing unit where the applications execute. A Router Manager is responsible for assigning the different sets of rules to the appropriate part (radio board, main CPU) and for delivering each packet to the appropriate application

All the routing protocols that need to maintain state for their proper functionality are implemented on the sensor node. In order to maintain this required state a routing agent should be executing on each node. The special functions that they need (e.g. CRC check, packet modification) can be provided by the first tier when the agents register with the Router Manager (Figure 2).

3. ANALYSIS AND MEASUREMENTS

In our prototype implementation an iKit2000 [7] development board is connected to a WINS sensor node [11] through a serial port (115.2Kbps). In our case the built-in radio was not used. Instead our prototype radio board on the iKit2000 was used. The iKit2000 has a Triscend E520 [10] chip, which is essentially a 8032 microcontroller core as well as 40K of FPGA. The development board is connected to an RFM [8] radio module (10Kbps, 256-byte packets), which is the actual radio hardware used. The current stage of prototype implementation includes software packet transmission and reception as well as the first stage of packet processing in the 8032 MCU. The second stage of packet processing is performed on the CPU of the WINS node. We are also in the process of building the hardware filters in the FPGA part of the Triscend E520 device.

Next, we present an analytical study and measurement data for the prototype implementation. Our analytical study considers both the incurred delay and the energy consumed by one packet.

Suppose that D_{RX}, D_{TX} are the delays to receive and transmit a packet for the software receiver/transmitter respectively. Assume for the MCU that D_{MCUFW} is the filter processing delay for packets that are going to be forwarded to other nodes, D_{MCUAC} is the filter processing delay for packets that are actually accepted by a node, and D_{SR} is the delay of the serial port between the radio board and the main node processor. Also assume that the

corresponding delays for the main node CPU are D_{CPUFW} and D_{CPUAC} respectively. Pr_{AC} and Pr_{FW} are the corresponding probabilities of a packet being accepted and being forwarded respectively. We don't consider the case of the dropped packets because most dropped packets are duplicate packets that are detected in the main sensor node and will cross the serial link anyway.

The delays without and with filtering in the MCU are:

$$D_{NF} = (D_{RX} + D_{SR} + D_{CPUAC}) \cdot \Pr_{AC} + (D_{RX} + D_{SR} + D_{CPUFW} + D_{SR} + D_{TX}) \cdot \Pr_{FW}$$

$$D_F = (D_{RX} + D_{MCUAC} + D_{SR}) \cdot \Pr_{AC} + (D_{RX} + D_{MCUFW} + D_{TX}) \cdot \Pr_{FW}$$

The difference in delays is:

$$D_{diff} = D_{NF} - D_F = (D_{CPUAC} - D_{MCUAC}) \cdot \Pr_{AC} + (2 \cdot D_{SR} + D_{CPUFW} - D_{MCUFW}) \cdot \Pr_{FW}$$

Our prototype implementation measurements are shown in Table 2. Given this data and packet acceptance and forwarding probabilities Pr_{AC}, Pr_{FW} the difference in delays is:

$$D_{diff} = -2.0285 \cdot \Pr_{AC} + 68.281 \cdot \Pr_{FW}$$

In order for this difference to be zero Pr_{FW} must be Pr_{FW}=0.0297 Pr_{AC} which is satisfied for the simulation data and for most practical sensor network protocols.

On the other hand the corresponding energy calculations are:

$$E_{NF} = (E_{RX} + E_{SR} + E_{CPUAC}) \cdot \Pr_{AC} + (E_{RX} + E_{SR} + E_{CPUFW} + E_{SR} + E_{TX}) \cdot \Pr_{FW}$$

$$E_F = (E_{RX} + E_{MCUAC} + E_{SR}) \cdot \Pr_{AC} + (E_{RX} + E_{MCUFW} + E_{TX}) \cdot \Pr_{FW}$$

And the difference in energy consumption is:

$$E_{diff} = E_{NF} - E_F = (E_{CPUAC} - E_{MCUAC}) \cdot \Pr_{AC} + (2 \cdot E_{SR} + E_{CPUFW} - E_{MCUFW}) \cdot \Pr_{FW}$$

If we assume that the power consumption of the main node CPU is α times the power consumption of the MCU and the serial port power consumption is the sum of the power consumption of the CPU and the MCU (because both devices should be on durinf serial port operation) then the difference in energy is:

$$E_{diff} = E_{NF} - E_F = (\alpha \cdot D_{CPUAC} - D_{MCUAC}) \cdot P_{MCU} \cdot \Pr_{AC} + (2 \cdot (\alpha + 1) \cdot D_{SR} + \alpha \cdot D_{CPUFW} - D_{MCUFW}) \cdot P_{MCU} \cdot \Pr_{FW}$$

Typical power consumption values for e.g. the Atmel AVR MCU and the E520 15mW and 470mw respectively. The actual power consumption of the WINS node is 351mW. This data result in two values of α: i) 23.4 for the WINS node/AVR combination and ii) 0.747 for the WINS node/E520 combination. For the two values of α the difference in energy is dominated by the $D_{SR,}$ which is essentially the penalty paid for crossing the boundary between the radio board and the CPU.

The value of α is expected to be much greater than one in most cases of sensor nodes although in one (bad) case above α is less than one. This is due to the fact that the E520 device hosts a 8032 MCU and a small amount of FPGA on the same die. However, for both values of α, the data on Table 1 and the probabilities of packets being accepted and forwarded (simulation data) we have values of E_{diff}, which are 1167 and 79 times the value of the MCU power consumption.

Table 1. Measured parameters on prototype

Parameter	Value(ms)
D_{MCUAC}	4.182
D_{MCUFW}	4.894
D_{CPUAC}	0.111
D_{CPUFW}	0.125
D_{SR}	36.532

4. CONCLUSIONS

As we have seen from the simulation data a considerable percentage of packets that enter a node are processed in a straightforward manner and are either redirected to the radio board, forwarded to the main processor or simply dropped. We proposed a two-tier architecture that enables the lower communication layers to perform the simple processing, drop or redirection of the packets as low as the radio board of a sensor node. As an additional feature, our architecture also enables the sensor applications to define methods for routing their own packets. We demonstrated a realization of the two-tier architecture in our prototype implementation, which includes a packet processor in the MCU of a system-on-a-chip.

5. ACKNOWLEDGEMENTS

This paper is based in part on research performed under DARPA Power Aware Computing and Communications program through AFRL contract # F30602-00-C-0154 and Sensor Information Technology program through AFRL contract # F30602-99-C-0128.

6. REFERENCES

[1] Chien C. et al, "Design experience with an integrated testbed for wireless multimedia computing", MoMuC '96, p.231-8.

[2] Estrin D., Govindan R., Heidemann J., Kumar S., "Scalable Coordination in Sensor Networks", MobiCOM '99, pp. 263-70.

[3] Johnson D., Maltz D., "Dynamic Source Routing in Ad Hoc Wireless Networks", Mobile Computing, pp. 153-181, Kluwer Academic Publishers, 1996.

[4] McCanne S., Jacobson V., "The BSD Packet Filter: a New Architecture for User-level Packet Capture", Proceedings of the Winter 1993 USENIX Conference, pp. .259-69, 1993.

[5] Atmel FPSLIC, http://www.atmel.com/atmel/products/

[6] Cots Dust, http://www-bsac.eecs.berkeley.edu/~pister/SmartDust/

[7] Ikit2000 Development Kit, http://www.ikit2000.com

[8] RF Monolithics, http://www.rfm.com

[9] Sensoria Corporation, http://www.sensoria.com

[10] Triscend Corp, http://www.triscend.com

[11] WINS Architecture, http://wins.rsc.rockwell.com