An Elaboration on Dynamically Re-configurable Communication Protocols Using Key Identifiers*

Kaushalya Premadasa and Björn Landfeldt

School of Information Technologies,
The University of Sydney,
Sydney 2006 Australia
{kpremada, bjornl}@cs.usyd.edu.au

Abstract. In this paper we elaborate on our novel concept and methodology for generating tailored communication protocols specific to an application's requirements and the operating environment for a mobile node roaming among different access networks within the global Internet. Since the scheme that we present employs a universal technique, it can be also deployed in small-scale independent networks such as sensor networks to generate application-specific lightweight transport protocols as is appropriate to its energy-constrained operating environments. Given that our proposed scheme is based on decomposing the communication protocols of the TCP/IP protocol suite, it allows sensor networks implementing the proposed scheme to easily connect to the existing Internet via a sink node consisting of a dual stack, without the loss of information in the protocol fields during the protocol translation process. We present preliminary experimental and analytical results that confirm and justify the feasibility of our method based on a practical example applicable to the sensor network environment.

1 Introduction

Networked and distributed computing is evolving at a tremendous rate. Both the applications and end hosts are becoming increasingly diversified and requirements on the behaviour and functions of the underlying communication infrastructure are following this evolution. The glue between the applications and the infrastructure is the communication protocol stack. In order to exploit the full potential of these diverse applications and nodes, the stacks should be configurable.

However, the current philosophy and design of the TCP/IP protocol stack has remained relatively intact for the past three decades and as a result, the individual protocols provide the same basic set of services to all applications regardless of individual needs.

In a networking environment such as wireless that grants the freedom of movement, the characteristics of the underlying network can change dynamically as a mobile node roams through different access networks thereby affecting the application's performance. As a result, we believe that the application requirements would also

* This research work is sponsored by National ICT Australia.

L.T. Yang et al. (Eds.): HPCC 2005, LNCS 3726, pp. 33 – 44, 2005.

change in order to maximize resource usage and throughput given the changed conditions.

Therefore, considering that both the application requirements and the network characteristics can change dynamically in a mobile computing environment, we need to be able to dynamically re-configure the communication protocols to suit the new operating environment in order to achieve the optimum results for the new residing network.

A configurable stack also offers a distinct advantage for thin clients such as sensors operating in energy-constrained wireless environments. Since the functions required for the applications or roles of such devices are known, it is possible to streamline the implementation of the communication stack to only implement these functions. This has the distinct advantage that power consumption can be minimized since only necessary computations have to be made. This in turn will translate to longer life span for battery-powered devices.

In [1] we presented our novel concept and methodology for generating dynamic communication protocols customized to an application's requirements and the network characteristics for a mobile node roaming through different access networks within the global Internet. In this paper we elaborate on the details of our proposed method and make the following contributions:

- We describe by way of an example how the proposed method can be also deployed in energy-constrained sensor networks for generating application-specific lightweight transport protocols due to the uniformity and universality of the presented scheme.
- We show that for our proposed scheme the computational requirement to implement sequence control functionality alone on a gateway consisting of a 32-bit Intel StrongARM SA-1110 processor is 217 clock cycles compared with TCP's 1679 clock cycles, given that TCP does not allow the provision to implement only the desired functions on need basis.

2 Related Work

The related work consists of two parts. Given that our proposed scheme is based on decomposing the communication protocols of the TCP/IP protocol suite, in the first part we describe previous work related to deployment of TCP/IP within sensor networks to enable seamless connectivity with the global Internet. In the second part we describe the related work as applicable to dynamically generated protocols.

2.1 Deployment of TCP/IP for Sensor Networks

Sensor networks require the ability to connect to external networks such as the Internet through which activities such as monitoring and controlling can take place. Deploying TCP/IP directly on sensor networks would enable seamless integration of these networks with the existing Internet. However, it had been the conventional belief that TCP/IP, the de-facto standard for the wired environment is unsuitable for the wireless sensor networks consisting of nodes of limited capabilities, since its implementation needs a large resource requirement both in terms of code size and memory usage.

[2] disproved this widely accepted norm by describing two small TCP/IP implementations for micro-sensor nodes such as motes consisting of 8-bit microcontrollers, without its implementations sacrificing any of TCP's mechanisms such as urgent data or congestion control. The proposed TCP/IP implementations were written independently from the Berkeley BSD TCP/IP implementation [3], since the BSD implementation was originally written for workstation-class machines and hence not catered for the limitations of small-embedded systems. On the other hand, InterNiche NicheStack [4] is a portable BSD-derived implementation of TCP/IP that can be deployed in a more high-end sensor such as a gateway consisting of a 32-bit microcontroller such as Intel's StrongARM SA-1110 processor.

The proposed implementations of TCP/IP above for the sensor nodes have contributed to address the problem of enabling seamless connectivity of sensor networks with the global Internet. However, because TCP does not allow the facility to selectively implement functions as needed, these implementations do not allow the generation of application-tailored lightweight transport protocols for the sensor networks. For instance, an application may desire to implement the transport functionalities of error detection and sequence control but without the cost of retransmissions. By using TCP, it is not possible to satisfy such a requirement.

Hence in this paper, we elaborate on the details of our proposed scheme for dynamically generated communication protocols that has the potential and capability to address this important issue. Also, our scheme can be applied for any TCP/IP implementation including those proposed for the sensor networks since it is based on decomposing the communication protocols of the TCP/IP protocol suite. Furthermore, as a result of this latter feature, it also allows sensor networks deployed with this scheme to easily connect to the existing Internet through a gateway, with a loss-free mapping of the transport protocol information in the protocol translation process.

2.2 Dynamically Generated Protocols

Some architectural principles for the generation of new protocols were presented in [5]. These included implementation optimization techniques such as Application Level Framing and Integrated Layer Processing to reduce interlayer ordering constraints. The concept of a "protocol environment" was introduced in [6] consisting of standard communication functionalities. It proposed the ability for an application to create flexible protocol stacks using standard protocol entities thus leading to the generation of application-tailored, extensible stacks. The Xpress Transfer Protocol (XTP) was defined in [7] that consisted of a selective functionality feature allowing the selection of certain functions such as checksum processing or error control on a per packet basis.

The x-Kernel [8] provided an architecture for constructing and composing network protocols and also simplified the process of implementing protocols in the kernel. The notion of adaptable protocols was proposed in [9] and presented the generation of flexible transport systems through the reuse of functional elements termed "micro-protocols". The DRoPS project in [10] was concerned with providing the infrastructure support for the efficient implementation, operation and re-configuration of adaptable protocols and DRoPs based communication systems were composed of micro-protocols.

All these approaches have contributed to advance knowledge and demonstrate the benefits of dynamically generated protocols. However, because of the complexity involved in parsing dynamically generated headers with varying formats and compositions these approaches, unlike our proposed approach have proven too complex to realize and be widely deployed.

3 The Proposed Concept

3.1 Concept of Generating Tailored Communication Protocols

The framework for our proposed work is a modified, five-layered OSI model consisting of the Application, Transport, Network, Data Link and Physical layers.

The central idea adopted in defining tailored communication protocols is the concept that the Transport and Network layers of the proposed model can be decomposed into separate functions. For instance, the Transport layer can be decomposed into end-to-end flow control, sequence control, error detection, error recovery etc. Hence based on the application's requirements and the network characteristics, the application has the ability to select the functions it wishes to implement from layers three and four of this model. As a result, a communication protocol is generated tailored to the specific needs consisting of the functional information belonging to the requested functions.

It should be noted that details relating to how an application's requirements and network characteristics are specified are beyond the scope of this paper considering that this is an Application Program Interface (API) design issue.

3.2 Use of Key Identifiers to Differentiate Among Functional Information

In order to overcome the previous difficulties of parsing dynamically generated headers and to efficiently recover the necessary functional information from the resulting communication header, in [1] we introduced the concept of "Key Identifiers" that will be used to differentiate among the functional data belonging to the various functions. The fundamental idea that forms the basis for this concept is that each function of the Transport and Network layers of the proposed model is assigned a unique key, a method used since the 1940's [11] in communication systems, most notably in Code Division Multiple Access (CDMA) systems.

On the transmitter side, to enable the process of recovering the required functional information efficiently, each individual functional information of the Dynamically Re-Configured Communication Protocol header is firstly multiplied by the unique key of the function to which it belongs and the individual results are then summed to produce the final resulting communication header that will then be transmitted along with the Application and Data Link layer headers and the payload.

The same key used for encoding a particular functional data field is used to decode the same at the receiver. The receiver simply multiplies the received communication header with the key to recover the information.

It is worth noting that although one is generally accustomed to associating the use of Keys in the context of security, in the scheme that we present the Keys are used for the purpose none other than allowing the transmission of any combination of func-

tional data as needed and efficient recovery of this information at a receiver. Therefore we derive a globally standardized key space for the Transport and Network layers of our proposed model such that each intermediate router and end-host maintains a table, mapping functions to keys so that the required information may be extracted.

A special key identification field is also included in the complete header that is transmitted in which each bit, relative to its position in this field, signifies whether the functional data belonging to a particular function is present as denoted by a "1" or conversely, the functional information is absent as denoted by a "0".

3.3 Process of Generating Key Identifiers

The chipping sequences that are used to encode the different functional information are the orthogonal Walsh functions. The main advantage of orthogonal codes is that they are able to completely eliminate multi-access interference as a result of their orthogonal property [12]. In the context of our work this translates to being able to correctly recover particular functional information from the received communication header, among the co-existence of many different functional data belonging to various functions in any given combination.

4 Key Distribution Approaches

Table 1 summarizes the fields that would be encoded and left un-encoded for the Transport and Network layers of the proposed model. The fields defined are derived from a composite of traditional communication protocols from the TCP/IP protocol suite.

Table 1. Summary of encoded/un-encoded fields

	Transport layer	Network layer
Encoded fields	Sequence #, Acknowledgement #, Source port #, Destination port #, Receive window, Internet checksum, Urgent data pointer, Flag Field (RST, SYN, FIN, ACK, URG, PSH, UNUSED, UNUSED)	Traffic class
Un-encoded fields		Destination address, Source address, Flow label, Hop limit

Given that the application-specific nature of sensor networks makes use of data-centric routing mechanisms as opposed to the Internet's address-centric mechanisms, only the functions applicable to transport layer will be implemented in the sensor nodes within these networks with the exception of the gateways.

There are two ways of assigning keys to fields and therefore also organizing the header information. In approach 1, each field is assigned a unique key to encode its functional information and in approach 2, the encoded fields from table 1 are grouped according to their bit lengths. Therefore, we derive three groups consisting of similar fields of 32, 16 and 8 bit lengths.

5 Mechanisms for Fast Computation

Given that computation in hardware is more efficient than in software, the multiplicative functions with respective keys would be implemented in hardware at the network layer instead of it being a full kernel implementation. Also, in order to allow fast and efficient recovery of desired functional information, an array of multipliers would be utilized to allow parallel multiplication of the received communication header by the appropriate key identifiers.

In the following section we present the initial experimental results for our proposed concept based on the mechanisms described in this section.

6 Experimental Results

6.1 Observed Results

It is of primary concern to investigate the computational overhead in using the key system compared with the standard way of sequentially extracting the header fields. We have therefore conducted a hardware simulation to determine the number of clock cycles it would take to decode a single bit that has been encoded by key lengths of 16-bits, 8-bits and 4-bits (as used by the key distribution approaches described in section 4) at an end-host, employing an array of multipliers operating in parallel as described in section 5.

The simulation was performed using Symphony EDA VHDL Simili, a software package consisting of a collection of tools that enable the design, development and verification of hardware models using VHDL, a hardware description language [13].

The experiment was conducted by encoding three bits (representing three separate bits from three different functional information) with three different keys that are generated using the key generation process described in section 3.5 above and then decoding the transmitted bits in parallel at the receiver (representing an end-host). Table 2 summarizes the results of this experiment.

Table 2. Results of the experiment

Key length (bits)	Number of clock cycles to decode each bit
16	10
8	6
4	4

Based on the results of the experiment given in table 2 above, table 3 summarizes the number of clock cycles it would take to decode a single functional information field in parallel for the proposed key distribution approaches described in section 4 above.

Table 3. Summary of clock cycles to decode a single field for the proposed key distribution approaches

	32-Bit field	16-Bit field	8-Bit field
Approach 1	32x10=320 clock cycles	16x10=160 clock cycles	8x10=80 clock cycles
Approach 2	32x4=128 clock cycles	16x6=96 clock cycles	8x4=32 clock cycles

6.2 Comparison with the Existing Method

The results of this experiment are very encouraging based on past research conducted to determine the overhead associated with TCP/IP protocol processing. In order to justify this conclusion, we firstly describe the past experiments that were conducted and present the results of those investigations. These results are then later used as a comparison point for evaluating the feasibility of our proposed method.

6.2.1 Related Work on TCP/IP Protocol Processing.

An experiment was conducted in [14] to analyze the TCP processing overhead for a modified Berkeley implementation of Unix on a 32-bit Intel 80386 processor with a clock speed of 16MHz and an instruction execution rate of 5 million instructions per second (MIPS). The TCP code was modified to give a better measure of the intrinsic costs involved. Fig.1 illustrates the methodology that was used for this analysis.

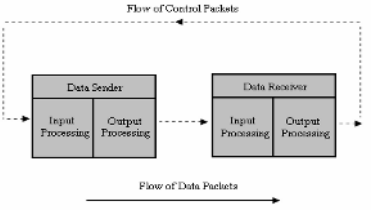

Fig. 1. Methodology used for analysis in [14]

It was discovered that for input processing at both a data sender and a data receiver the common TCP protocol-specific processing path consisted of 154 instructions of which 15 were either procedure entry and exit or initialization. In particular, the input processing for a data receiver also consisted of an additional 15 instructions for sequencing the data and calling the buffer manager, 17 instructions for processing the window field of a packet and 25 instructions for finding the Transmission Control Block (TCB) for the TCP connection. It is worthwhile mentioning that the reported instruction counts for various TCP protocol-specific tasks consisted of extracting the

information from the header fields and execution of the corresponding algorithms. It was also found that the output processing of a data receiver consisted of a total of 235 instructions to send a packet in TCP. Therefore based on these results, for a data receiver, the ratio of input to output processing instructions is given by 211:235 respectively.

Although all the reported experiments conducted in [14] were on processors of a Complex Instruction Set Computer (CISC) architecture, the authors have stated that based on a separate study of packet processing code, they found little expansion of the code when converted to a Reduced Instruction Set Computer (RISC) chip. They concluded that this is because given the simplicity of the operations required for packet processing, irrespective of the processor that is used the instruction set actually utilized is a RISC set. Given that the authors have estimated the ratio of CISC to RISC instructions to be approximately 3:4 respectively, 211 and 235 instructions for an 80386 processor would be translated to 282 and 314 instructions respectively for a RISC processor.

In an independent experiment also conducted in [14] to measure the actual costs involved with a Berkeley TCP running on a Sun-3/60 workstation with a 32-bit Motorola 68020 processor with a clock speed of 20MHz and an instruction execution rate of 2 MIPS it was discovered that it took 370 instructions for TCP checksum computation which is the major overhead associated with TCP processing. This figure thus translates to 494 instructions for a RISC processor based on the above ratio of CISC to RISC instructions.

In a similar study conducted in [15] an experiment was conducted to determine the overhead associated with the Ultrix 4.2a implementation of TCP/IP software processing. The experiment consisted of one workstation sending a message to the system under test (where all measurements were taken) which then sends the same message back to the originator. All measurements were taken on a DECstation 5000/200 workstation, a 19.5 SPECint MIPS RISC machine with a clock speed of 25MHz and an instruction execution rate of 20 MIPS.

In this experiment it was discovered that for TCP messages, protocol-specific processing (that includes operations such as setting header fields and maintaining protocol state with the exception of checksum computations) consumes nearly half the total processing overhead time. It was also discovered that TCP protocol-specific processing in particular dominates the protocol-specific processing category consuming approximately 44% of the processing time. The TCP protocol-specific processing however did not include checksum computation in its classification.

In this experiment it was discovered that 3000 instructions were consumed for TCP protocol-specific processing at a data receiver. Based on the ratio of RISC processor instructions for input to output processing for a data receiver in [14], approximately 1420 instructions were thus consumed in [15] for Input processing.

6.2.2 Evaluation of the Feasibility of the Proposed Method

In order to demonstrate the viability of our proposed method we present a practical example within the sensor network environment in conjunction with the routing protocols, in which the transport layer functionality of sequence control would be very desirable. We confirm the feasibility of our approach by evaluating the number of clock cycles that would be consumed to implement this functionality alone with our

proposed method to that consumed using the traditional sequencing transport protocol TCP that does not allow the provision to implement only the desired functions on need basis.

There are no data available in relation to the processing times for the implementations of TCP/IP as described in section 2.1 for the sensor networks. However, given that InterNiche NicheStack as reported above consists of a BSD-derived implementation of TCP/IP that can be deployed in a more high-end sensor such as a gateway consisting of a 32-bit microcontroller, we therefore believe that it is fair to assume that its implementation would be very similar to those described in section 6.2.1, and we conduct our analysis based on this assumption.

In [16] a family of adaptive protocols called Sensor Protocols for Information via Negotiation (SPIN) was proposed for the Network layer for disseminating information among sensor nodes in an efficient manner. The goal of SPIN family of protocols is to conserve energy through negotiation by firstly communicating a high-level data descriptor called a meta-data that describes the data without transmitting all the data. SPIN protocols employ a simple three-way handshake mechanism for negotiation based on three types of messages: ADV, REQ and DATA. Prior to sending the DATA message, the sensor node broadcasts an ADV message containing the meta-data. Neighbours that are interested in the data then respond with a REQ message after which the DATA is sent to those sensor nodes. This process is thus repeated by the neighbour sensor nodes once they receive a copy of the interested data. Therefore, all interested nodes within the entire sensor network would eventually receive a copy of this data.

For a sensor network employing such a data transaction mechanism, it would be very desirable to be able to utilize the transport layer functionality of sequence control to allow the data receiving neighbour nodes to sequence the data, at the cost of minimum overhead. Using our proposed methodology this can be very easily fulfilled by appending to the broadcasted ADV message a dynamically generated communication header consisting of the encoded sequence number field bootstrapped with the chosen initial sequence number. As a result, the neighbours that respond with a REQ message would be all aware of the initial sequence number associated with the data transaction which is to follow. Thereafter, these nodes can use the information in the encoded sequence number field of subsequent data packets to organize the received data in their proper order.

In [14] it was discovered that it took 15 instructions for sequencing the data and calling the buffer manager that translates to 20 RISC processor instructions, based on the ratio of CISC to RISC instructions reported above. Therefore based on the ratio of RISC processor instructions for the sequence control functionality to that of input processing for a data receiver in [14], approximately 101 instructions are thus consumed in [15] for the same sequence control functionality.

Table 4 provides a summary of the instruction counts for various tasks as reported above and calculates the number of clock cycles it takes to execute these instructions on a gateway to a collection of motes consisting of a 32-bit Intel StrongARM RISC SA-1110 processor with a clock speed of 206 MHz and an instruction execution rate of 235 MIPS [17]. In this analysis we make the assumption that the instruction count for TCP checksum computation based on a RISC processor in [14] is similar for the same in [15].

Table 4. Summary of instruction counts for various tasks and the associated clock cycles for their execution

	Instruction count as per [15]	Number of clock cycles as per StrongARM SA-1110 processor
Sequencing data	101	89
Input processing at data receiver	1420	1245
TCP checksum computation	494	434

In Table 5 and in fig.2 we therefore provide a comparison of the number of clock cycles it takes on the StrongARM SA-1110 processor to implement the sequence control functionality alone with our proposed methodology to that of the traditional sequencing transport protocol TCP that does not allow the facility to implement only the desired functions on need basis. In TCP it is also not possible to disable certain functionality by simply setting a particular header field to zero since for most fields zero is a valid value.

For the results of table 5 below, it should be noted that in the case of our proposed method based on both key distribution approaches, during the time spent to decode the sequence number field, up to an additional 12 and 14 encoded functional fields can also be decoded in parallel at a node for key distribution approaches 2 and 1 respectively. Also, the value of 89 clock cycles for sequencing the data actually comprises of the cycles consumed for both extracting the information from the header field and execution of the corresponding algorithm associated with the sequence control function. Therefore, the total cycles consumed for implementing the sequence control functionality using our method would be theoretically less for both key distribution approaches than the calculated values if the cycles consumed for extracting the functional information from the header field had been subtracted.

Table 5. Comparison of clock cycles for sequencing of data for our proposed method against TCP

	Number of clk cycles for our method: key distribution approach 2	Number of clk cycles for our method: key distribution approach 1	Number of clk cycles for TCP
Decoding sequence number field	128	320	
Sequencing data	89	89	
Input processing at data receiver			1245
TCP checksum computation			434
TOTAL	217	409	1679

From the results of fig.2 below it can be clearly seen that the number of clock cycles consumed for sequencing of data using our method based on key distribution

approach 2 and approach 1 are 1462 and 1270 cycles respectively less than that for the case of TCP. Also key distribution approach 2 consumes only about half the number of clock cycles compared to key distribution approach 1. Therefore through this simple but practical example we have demonstrated the feasibility and viability of our proposed method for tailoring communication protocols for the sensor network environment for which energy conservation is of utmost importance.

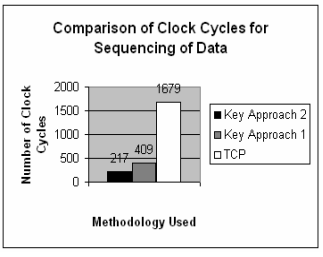

Fig. 2. Comparison of clock cycles for sequencing of data for our proposed method against TCP

7 Conclusions and Future Work

In this paper we have elaborated on the details of our novel method based on the well-known CDMA technique for dynamically generating communication protocols customized to an application's requirements and the network characteristics within the wireless environment. The advantages of being able to dynamically generate communication protocols based on a specification have been made clear, with its benefits extending to a spectrum of wireless devices with varying processing capabilities such as sensor nodes and portable laptops.

The feasibility and viability of the proposed method have been proven with initial experimental work and through comparison of these results to those obtained from past studies carried out to discover the cost of TCP/IP protocol processing overheads.

We are currently working on a header compression technique for the proposed scheme. As continuing work, we will be working toward a full system implementation of the proposed concept for the sensor networks.

References

1. Premadasa, K., Landfeldt, B.: Dynamically Re-Configurable Communication Protocols using Key Identifiers. In: to be published in the proceedings of ACM/IEEE 2nd conference on MobiQuitous 2005, San Diego, California, USA (July 2005)
2. Dunkels, A.: Full TCP/IP for 8-bit architectures. In: Poc. 1st conference on MOBISYS'03 (May 2003)
3. McKusick, M.K., Bostic, K., Karels, M.J., Quarterman, J.S.: The Design and Implementation of the 4.4 BSD Operating System. Addison Wesley, United States of America (1996)

4. InterNiche Technologies Inc NicheStack portable TCP/IP stack: www.iniche.com/products/tcpip.htm
5. Clark, D.D., Tennenhouse, D.L.: Architectural Considerations for a new Generation of Protocols. In: ACM SIGCOMM Computer Communications Review, Vol. 20, No.4 (August 1990) 200-208
6. Tschudin, C.: Flexible Protocol Stacks. In: ACM SIGCOMM Computer Communications Review, Vol. 21, No.4 (August 1991) 197-205
7. Strayer, W.T., Dempsey, B.J., Weaver, A.C.: XTP: The Xpress Transfer Protocol. Addison Wesley, United States of America (1992)
8. Hutchinson, N.C., Peterson, L.L.: The x-Kernel: An architecture for Implementing Network Protocols. IEEE Transactions on Software Engineering, Vol. 17, No.1 (January 1991) 64-76
9. Zitterbart, M., Stiller, B., Tantawy, A.: A Model for Flexible High-Performance Communication Subsystems. In: IEEE Journal on Selected Areas in Communications, Vol. 11, No.4 (May 1993) 507-518
10. Fish, R.S., Graham, J.M., Loader, R.J.: DRoPS: Kernel Support for Runtime Adaptable Protocols. In: Proc. 24th Euromicro conference '98, Vol.2, Vasteras, Sweden (1998) 1029-1036
11. Scholtz,: The Evolution of Spread-Spectrum Multiple Access Communications. In: Code Division Multiple Access Communications. Glisic, S.G., Leppanen, P.A. (eds.): Kluwer Academic Publishers (1995)
12. Rhee Y.M.: CDMA Cellular Mobile Communications & Network Security. Prentice Hall Inc., United States of America (1998)
13. Symphony EDA. URL: http://www.symphonyeda.com
14. Clark, D., Jacobson, V., Romkey, J., Salwen, H.: An Analysis of TCP Processing Overhead. In: IEEE Communications Magazine, 50th Anniversary Commemorative Issue (May 2002) 94-101
15. Kay, J., Pasquale, J.: Profiling and Reducing Processing Overheads in TCP/IP. In: IEEE/ACM Transactions on Networking, Vol. 4, No.6 (December 1996) 817-828
16. Heinzelman, W.R., Kulik, J., Balakrishnan, H.: Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In.: Proc. ACM Mobicom '99, Seattle, Washington, USA (1999) 174-185
17. Intel StrongARM SA-1110 Datasheet. URL: www.intel.com/design/strong/datashts/278241.htm