Controller Network Data Extracting Protocol – Design and Implementation

Nikolay Kakanakov, Ivan Stankov, Mitko Shopov, Grisha Spasov

Abstract: *The paper presents the design and implementation of a UDP-based protocol for Distributed Automation Systems. It is based on client/server interactions. Protocol specification is given together with its syntax, grammar and semantics. Message formats, protocol vocabulary and communication rules are described. The possible applications of the protocol are discussed and a sample implementation of the server and client are shown in the paper. Initial tests of the effectiveness of the protocol are made. The experiments are test-bed, carried out in the experimental network in "Distributed Systems and Computer Networks Lab" in Technical University of Sofia, branch Plovdiv (http://net-lab.tu-plovdiv.bg/). They include evaluation of the communication capacity of the protocol. The minimum, maximum and average response times are calculated from the experimental results.*

Key words: *Embedded Networking, Protocol Design and Implementation, Distributed Automation.*

INTRODUCTION

Over the recent years there is a trend for adaptation of enterprise technologies in Distributed Automation Systems. The vendor specific standards are replaced with popular open standards of communication between automation nodes in plants, and between plants themselves. Most automation standards are limited in distance, number of devices or speed. This leads to implementations of automation protocols based on standard TCP/IP communication over standard links (Ethernet, WiFi, ATM) [1, 5, 6].

The paper deals with implementation and design of UDP-based protocol for communication with automation nodes in a plant.

Background

A protocol is a kind of agreement about the exchange of information in a distributed system. It defines a precise format for valid messages (syntax); the procedure rules for data exchange (grammar); and a vocabulary of valid messages that can be exchanged with their meaning (semantics). In a way, it formalizes the interaction by standardizing the use of a communication channel. The protocol, then, can contain agreements on the methods used for [2, 4]:

- Initiation and termination of protocol data units (message);
- Formatting and encoding data.
- Synchronization of senders and receivers;
- Detection and correction of transmission errors;

The widespread use and expansion of communications protocols is both a prerequisite to the Internet, and a major contributor to its power and success. Object-oriented programming has extended the use of the term to include the programming protocols available for connections and communication between objects [2].

Generally, only the simplest protocols are used alone. Most protocols, especially in the context of networking, are layered together into protocol stacks where the various tasks listed above are divided among different protocols in the stack [2].

A protocol specification consists of five distinct parts. To be completed, each specification should include explicitly [2]:

- The service to be provided by the protocol
- The assumptions about the environment in which the protocol is executed
- The vocabulary of messages used to implement the protocol
- The encoding (format) of each message in the vocabulary
- The procedure rules guarding the consistency of message exchanges

These five parts for CNDEP specification are described in the following section.

CONTROLLER NETWORK DATA EXTRACTING PROTOCOL

CNDEP works on application layer of TCP/IP protocol stack (figure 1). As a transport protocol it uses UDP. An area where UDP is especially useful is the client-server applications – this is the CNDEP base. The client sends a short request to the server and expects a short reply back. If either the request or reply is lost, the client can just time out and try again. CNDEP mechanism is replying back a suitable message – a time out for instance, to the user or any application. In this case not only is the code simple, but fewer messages are required (one in each direction) than with a protocol requiring an initial setup.

The reasons for UDP using are: short message exchange; fast communication; good reusability of communication channel; service of devices in sensible time [4].

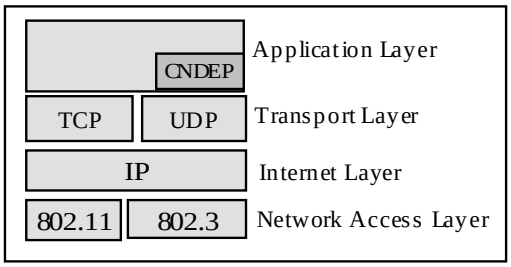

Figure 1: The place of CNDEP in TCP/IP stack.

Design principles have been applied to create a communication protocol over wide spread regular TCP/IP networks. These principles include effectiveness, reliability, and resiliency. All efforts are aimed to reach these features in most appropriate way [4].

Effectiveness – The particular reasons CNDEP is believed to be effective: fast transmissions; reliable of data exchange; computing power; working environment;

Reliability – Assuring reliability of data transmission involves error detection and correction, or some means of requesting retransmission. These functions are divided between Data Link layer and our application (described below).

Resiliency – it addresses a form of network failure known as topological failure in which a communications link is cut, or degrades below usable quality. These functions are inherited from the network layer (protocol IP).

The main purpose of CNDEP is for data extraction. Extraction could be done from any device that implements server side of CNDEP. Server side extracts data from sensors, for instance, and waits for any request to send back a suitable respond.

The environment of the protocol is defined from Local Area Networks. The idea of CNDEP is for data extraction in networks where data producing layer is situated behind a server. This part of a protocol description is consistent with direct relation to Protocol Service Specification. All assumption made for LAN and all advantages of client/server architecture are in present for protocol environment.

The data flow is from client to server. Client application makes all necessary data processing over received response. It is supposed that the client application is working on machine with better computing capabilities than the server application. Time sequence diagram of possible interactions is shown on figure 2.

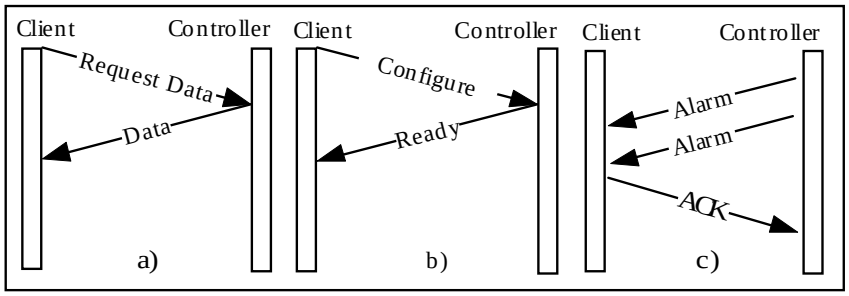

Figure 2: Protocol interactions.

The protocol vocabulary defines three distinct types of messages: *SET* for a message setting parameters, combined with a positive or negative acknowledgment; *GET* for an extracting message, combined with a positive or negative acknowledgment, and *TEST* messages for keep-alive transmission. The vocabulary can be succinctly expressed by set equations - (1), (2) and (3):

$$V_{REQUEST} = \{TEST, GET, SET\} \quad (1)$$

$$V_{RESPONSE} = \{OK, ERROR, DATA\} \quad (2)$$

$$V_{PROTOCOL} = (V_{REQUEST} \times V_{RESPONSE}) / (TEST, DATA) \quad (GET, OK) \quad (SET, DATA) \quad (3)$$

The CNDEP message consists of start byte, end byte and data fields. The data fields are used to encode the sessions, command, function, response type and actual data. Structure of the message is shown on figure 3.

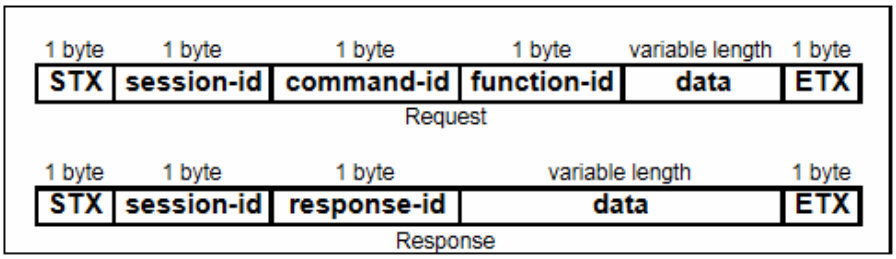

Figure 3: CNDEP messages.

CNDEP uses ASCII symbols for start and end of messages, respectively 02 (STX) and 03 (ETX). The exchanged messages are byte oriented. When a request is received by any device, the command is extracted from the message and is parsed. Each message type can further be refined into a class of lower-level messages, consisting for instance of one sub-type for each character code to be transmitted – figure 4.

$$Message ::= \{\langle \mathrm{Re}\, quest\rangle \mid \langle \mathrm{Re}\, sponse\rangle\};$$
$$\mathrm{Re}\, quest ::= STX\ \langle Session\_ID\rangle \langle Command\_ID\rangle [\langle Function\_ID\rangle][\langle Data\rangle] ETX;$$
$$\mathrm{Re}\, sponse ::= STX\ \langle Session\_ID\rangle \langle \mathrm{Re}\, sponse\_ID\rangle [\langle Context-Type\rangle][\langle Data\rangle] ETX;$$

Figure 4: BNF description of CNDEP messages.

Session_ID ::= Byte, representing the session (for retransmission).
Command_ID ::= Byte, representing the command to controller.
Function_ID ::= Byte, showing subcommand (if any).
Response_ID ::= Byte, representing the response type.
Context-Type ::= MIME Types – describes the type of the data field.
Data ::= Variable Length String.

The exact value of each field is extracted after obligatory parsing. Parsing is needed for recognizing what function to be executed by the server (device). The first one is describing the user session that represents Request/Respond pair. Command-id shows the command number as a consequence number in the list of all commands (Table 1). This field is used through the time of protocol developing. The function-id field comprises the real tasks that have to be done as sub-function of some command. Data fields are character oriented – strings with different length.

Table 1: Some CNDEP commands

		GET			SET			
Byte value	0	1	2	122	129	253	254	255
Command name	Test (OK)	Temperature	Humidity	User Data	User Data	Temperature Options	Humidity Options	Test (Error)

Since CNDEP is an asymmetric protocol, it has different function in the client and server parts. Assuming that the server runs on an embedded device with restricted

resources, its functions are design related only to the automation. Session logs, lost of packets and retransmissions are delegated to the client.

Server application accepts and parses requests made by the clients. It recognizes commands requested and contacts the appropriate sensor or actuator to perform the action. State transition diagram of the server's stub is given on Figure 5.

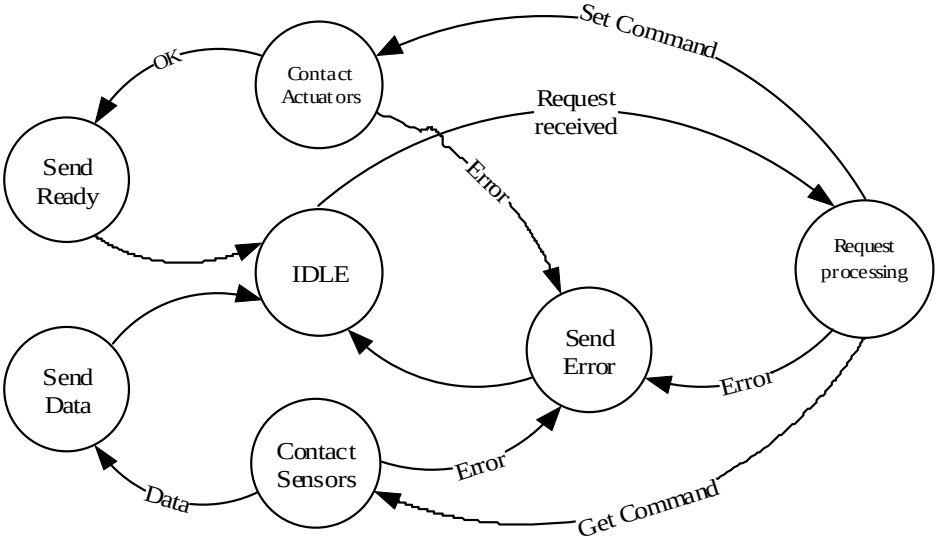

Figure 5: State Transition Diagram of server's application.

Clients track sessions (which are actually request-response pairs) in order to recognize which response corresponds to a given request. The session is used in error checking in order client to find duplicated packets in case of retransmission.

If error occurs in transmission, client application has the opportunity to insist a retransmission. Retransmission is optional feature. The ideology is not reliable communication but last-is-best. The latest data returned from the controllers is preferred because it is more actualize.

The Controllers did not have to track duplication of requests. Two subsequent identical requests will lead to two execution of the command in the request and therefore to two identical responses. There is no actual problem of duplicated execution. The duplicated responses will be noticed by the client application and it will decide which of them to use. In the simple case duplicated responses will be discarded, but in some occasions the latest arrived packet could overwrite the old one. The algorithm of the client application is shown on figure 6.

```
1.  Send Request;
2.  If (OK or DATA not received for TIME_OUT ms)
            REPEAT - -;
    Else goto 12;
3.  If (REPEAT <= 0)
            goto 11;
4.  Generate random number T in [MIN_DELAY .. MAX_DELAY];
5.  Wait T milliseconds;
6.  Send Request;
7.  If (OK or DATA not received for TIME_OUT ms)
            REPEAT - -;
    Else goto 12;
8.  If REPEAT <= 0 goto 11;
9.  T=T*2;
    If (T > UPPER_DELAY)
    T = UPPER_DELAY;
10. goto 5;
11. Error – Destination unreachable in REPEAT retries;
12. Done.
```

Figure 6: Example CNDEP client application.

Experimental analysis of the implementation

After a particular protocol is designed and implemented it must be validated. Validation can be done by simulation, test-bed experiments or real-word deployment.

Initial validation chosen for CNDEP is test-bed experiments and estimation of request-response times of some of the CNDEP commands. The experiments are carried out in "Distributed Systems and Computer Networks Lab" in Technical University of Sofia, branch Plovdiv [7]. The experimental network consists of embedded systems and a client. The embedded systems involved are DS TINI [8] and IPC@Chip [9]. The client is a PC running Windows XP. The nodes are connected in switched Ethernet network, working on 100Mbps.

The estimation is made for one command of each type – TEST, GET and SET. The estimated command is executed repeatedly a number of times in equal intervals. The graphical representation of the results for TEST command is shown on figure 7.

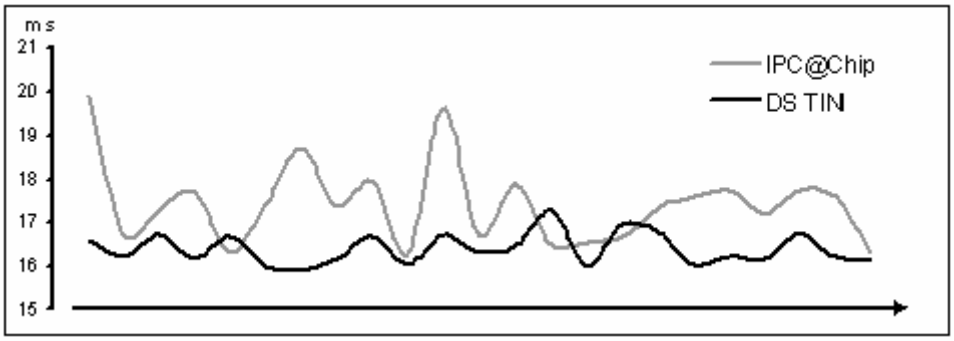

Figure 7: Request/Response times of TEST command for two different embedded systems.

The experimental results are used for calculating the minimum, maximum and average values of the request/response times per command, per embedded system. The calculated values in milliseconds for DS TINI [8] and IPC@Chip [9] are shown in Table 2.

Table 2: CNDEP Request/Response times

command / controller	TEST			GET			SET		
	min	max	avg ± σ	min	max	avg ± σ	min	max	avg ± σ
IPC@Chip	16.31	19.89	17.43 ± 0.20	334.76	338.25	336.38 ± 0.22	41.69	43.77	42.68 ± 0.13
DS TINI	15.91	17.27	16.4 ± 0.08	518.73	524.16	520.77 ± 0.31	55.76	59.31	57.04 ± 0.26

The relatively big values for the GET command can be explained by the fact that the time for contacting the sensor to get temperature or humidity is about 250ms.

CONCLUSIONS AND FUTURE WORK

The presented protocol is suitable for developing Distributed Automation Systems in TCP/IP environment. It is a custom application layer protocol implemented over the standard protocols stack, which makes it applicable for accessing data from the networked embedded devices.

The promoted results show that the response time has small deviation from its mean. It is very important for soft real-time embedded devices to have relatively constant delays, which means predictability.

In [3] request/response times of echo protocol for embedded devices are presented. The difference in request/response times of the echo protocol and CNDEP allows separating protocol delay from TCP/IP stack delay. Comparison between the two protocols shows that CNDEP did not lead to significant delays of communication.

Further, the simulation analysis of the protocol effectiveness (calculation of maximum nodes that can work together in a single network using CNDEP; comparison of CNDEP implementation over TCP and UDP; protocol traffic measurement; etc.) and development (adding new commands; developing version of CNDEP for wide area networks; etc.) should be done.

ACKNOWLEDGEMENTS

The presented work is supported by National Science Fund of Bulgaria project – **"ВУ-966/2005**", entitled "Web Services and Data Integration in Distributed Automation and Information Systems in Internet Environment", under the contract "**ВУ-МИ-108/2005".**

REFERENCES

[1] Estrin, D., G. Borriello, R. Colwell, J. Fiddler, M. Horowitz, W. Kaiser, N. Leveson, B. Liskov, P. Lucas, D. Maher, P. Mankiewich, R. Taylor, J. Waldo, Embedded, Everywhere. A Research Agenda for Networked Systems of Embedded Computers, NAP, Washinton, D.C. 2001, ISBN 0-309-07568-8.

[2] Holzmann, G., Design and Validation of Computer Protocols, Prentice Hall, 1991, ISBN: 0-13-539925-4.

[3] Kakanakov, N., Experimental Analysis of Client/Server Applications in Embedded Systems, proceedings of ELECTRONICS'05, 21-23 Sept.2005, Sozopol, Bulgaria, book 4, pp 97-102, ISBN:954-438-520-7.

[4] Stallings, W., High-Speed Networks and Internets, 2nd Ed, Prentice Hall, 2002, ISBN: 0-13-032221-0.

[5] Topp Topp, U., P. Müller. Web based service for embedded devices. Lecture Notes in Computer Science, Volume 2593 / 2003, pp. 141 – 153, ISSN: 0302-9743.

[6] Youngblood, G. M., Smart Environments, Ch. 5: "Middleware", pp. 101-127, 2004., ISBN: 0-471-54448-5.

[7] http://net-lab.tu-plovdiv.bg/ - Laboratory of Distributed Systems and Computer Networks.

[8] http://www.maxim-ic.com/products/tini/ - DS TINI Homepage.

[9] http://www.beck-ipc.com/ipc/ - IPC@Chip Homepage.

ABOUT THE AUTHORS

Nikolay Kakanakov, PhD Student, Department of Computer Systems and Technologies, Technical University of Sofia, branch Plovdiv, Phone: +358 32 659 758, e-mail: kakanak@tu-plovdiv.bg.

Ivan Stankov, BSc graduate student, Department of Computer Systems and Technologies, Technical University of Sofia, branch Plovdiv, Phone: +358 32 659 758, e-mail: istankov@tu-plovdiv.bg.

Mitko Shopov, BSc graduate student, Department of Computer Systems and Technologies, Technical University of Sofia, branch Plovdiv, Phone: +358 32 659 758, e-mail: mshopov@tu-plovdiv.bg.

Assoc. Prof. Grisha Spasov, PhD, Department of Computer Systems and Technologies, Technical University of Sofia, branch Plovdiv, Phone: +358 32 659 724, e-mail: gvs@tu-plovdiv.bg.