A Redundant Dynamic Host Configuration Protocol for Collaborating Embedded Systems

M. Schukat, M.P. Cullen, and D. O'Beirne

***Abstract*—**This paper describes a UDP over IP based, server-oriented redundant host configuration protocol (RHCP) that can be used by collaborating embedded systems in an ad-hoc network to acquire a dynamic IP address. The service is provided by a single network device at a time and will be dynamically reassigned to one of the other network clients if the primary provider fails. The protocol also allows all participating clients to monitor the dynamic makeup of the network over time. So far the algorithm has been implemented and tested on an 8-bit embedded system architecture with a 10Mbit Ethernet interface.

***Keywords*—**Ad-Hoc Networks, Collaborating Embedded Systems, Dynamic Host Configuration, Redundancy.

I. Introduction

CONSIDER a secured ad-hoc network consisting of a number of collaborating embedded systems (CES) that communicate via a TCP/IP protocol stack. The network is supposed to provide the following network services:

- *Authentication* of newly attached network clients.
- *Configuration* of authenticated network clients.
- *Propagation* of available services that are offered by individual clients.
- *Secure* point-to-point and broadcast/multicast data communication between clients.
- *Monitoring/logging* of client and network configurations.

All these services must the highly available and require therefore a redundant implementation.

Examples for such a network are a team of cooperative robots that explore an unknown terrain or a set of wireless clients that form a secure and highly available ad-hoc network.

The focus of this paper is on configuration services and their redundant implementation, as well as on methods to monitor the dynamic make-up of an ad-hoc network.

Configuration services under TCP/IP provide network clients with a unique (static or dynamic) IPv4 network address, a netmask and optional parameters (like the IP address of a DNS server). The tracking of the assignment of dynamic IP addresses over time is a desired monitoring feature that can be used to analyse collaboration strategies of robots or to fulfill liability issues of the participating clients in an ad-hoc network.

There are both server-oriented and server-less configuration protocols defined. This paper will show that they provide either enhanced availability or simple monitoring capabilities, but not both. Consequently, a new configuration protocol will be defined that fulfills both required features. The protocol is loosely related to DHCP and can be easily implemented on an embedded system with limited resources.

Manuscript received November 19, 2004.

Dr. Michael Schukat is a lecturer with the Department of Information Technology, National University of Ireland, Galway (phone: +353-91-512419; fax: +353-91-750501; e-mail: michael.schukat@nuigalway.ie).

Michael P. Cullen is a postgraduate student with the Department of Information Technology, National University of Ireland, Galway (email: michael.cullen@nuigalway.ie)

Declan O'Beirne is a postgraduate student with the Department of Information Technology, National University of Ireland, Galway (email: declan.obeirne@geminga.it.nuigalway.ie)

II. State of the Art

A. Server-Oriented Protocols

DHCP (Dynamic Host Configuration Protocol) is a TCP/IP service protocol that provides a mechanism for automating the configuration of network devices [1]. The specification of the base protocol can be found in RFC 2131 and RFC 2132.

Whenever a client physically joins a network, it sends a UDP broadcast request to a DHCP server, which in turn provides the client via a UDP broadcast response message with a unique IP address, a netmask and other parameters. Once a client has accepted an offer by the DHCP server by returning a response (followed by an acknowledgement by the server), it has to renew its lease periodically.

A DHCP server keeps a list with all allocated IP addresses, the associated MAC addresses and the lease times. This information can easily be logged solely by the server, but not by other network clients.

DHCP based on a single server does not provide the required high-availability and will fail if the disabled network-client runs the DHCP service. As a consequence newly attached clients cannot get assigned an IP address, and existing clients cannot renew their lease.

DHCP principally allows for multiple DHCP servers in a network, whereby a client might get multiple offers after an address request and has to make a choice. But there is no mechanism defined through which servers can communicate with each other to keep their databases of assigned IP addresses synchronised. Without an inter-server protocol, servers are constrained to assign and renew addresses from disjoint pools of available addresses. There is no redundancy mechanism that allows one server to renew a license that has been assigned by a different server before.

Cisco Systems have implemented a failover protocol for DHCP that is based on a primary and a secondary server [2]-[3]. In normal operation, the primary server provides host configuration services. It also sends a record of every transaction to the secondary server. If the primary server fails, the secondary server resumes its work, until the primary server is operational again. Before the primary server takes over again, it gets a list of transactions the secondary server has performed during its downtime.

Although the Cisco implementation supports redundancy, it is obvious that this protocol will fail if both the primary and the secondary server are being removed from the system, as it might happen in an ad-hoc network. Consequently, a fully redundant host configuration protocol requires that every client is a potential candidate to run this service.

B. Server-Less Implementations

There are a number of server-less protocols defined that allow the dynamic assignment of an IP address without the intervention of a DHCP server.

The IPv4 link-local autoconfiguration specification (IPv4LL) by the Internet-Engineering Task Force (IETF) Zeroconf Working Group describes a protocol that enables clients to choose their IP address autonomously within the 169.254 / 16 IP address block [4]. It is widely accepted by the software industry and currently being implemented in various Apple, Windows and Unix operating systems.

With IPv4LL a client that is attached to a network chooses a random IP address within the 169.254 / 16 range and verifies via ARP, if another client already uses the address. Depending on the response it gets it verifies either a new random address or uses the chosen one.

Another approach, employed in the context of IPv6 stateless autoconfiguration, uses the MAC-address of the host as a base to generate a unique IP address [5].

In the context of mobile ad-hoc networks (MANETs), a number of autoconfiguration protocols have been drafted [6] – [7]. MANETs are more complex than cable-based networks that are considered by IPv4LL and require therefore a more sophisticated IP address verification scheme.

All approaches provide a highly available configuration service, since every client has to configure itself. Since none of the clients keep track of the entire network configuration, a monitoring feature cannot be provided.

III. The Redundant Host Configuration Protocol

A. Overview

The proposed protocol provides both service redundancy and a monitoring feature for all interested network clients. The service is provided by one client at a time and runs concurrent to other client-tasks. Since there are a number of candidates that could run the service, clients have to compete against each other to select the service-providing client. This implies that all clients within the ad-hoc network trust each other, making an authentication service as mentioned in Section I necessary.

The service is based on the connection-less transport-layer protocol UDP using the sending port 4440 and the receiving port 4442, whereby all packets are broadcasted using the IP address 255.255.255.255. In its current implementation the protocol is supposed to work on a single-segment network providing 255 different dynamic IP addresses in the 192.168.n / 8 range. Relay agents as used in DHCP for multi-segment coverage are not supported.

The main idea behind this algorithm is that one active RHCP server manages all IP-address requests at a time. Whenever a new IP address has been assigned, the server updates its internal address table and distributes the updated table via broadcasting to the other clients. Address leases do not expire and it is up to the server to verify that a client with a specific IP address is still attached to the network. Clients do not give up their IP-address when detached due to the ad-hoc nature of the network. Whenever a server is not reachable any more, the residual network devices negotiate, which one will take over to run the service. This selection process depends on the age of the address table stored in each client and the willingness of each client to run the service. Clients with outdated address tables are not taken into consideration.

The diagram in Figure 1 shows the various states a network client can be in:

- Whenever it joins the network it enters the *INIT* state, where a previous assigned IP address is discarded.
- From there on it moves unconditionally to the *QUERY* state, where it sends a broadcast UPD GET_IP request in order to get an IP address. The currently active RHCP server receives the request and selects a free address, which is then marked as used. The IP address table is updated and the MAC address stored. The server then returns a SET_IP packet via broadcast, which will be specifically read by the client to set its own IP address, but which will also be used by the other clients to update their copy of the address table.
- If the server cannot provide an IP address, it returns a NO_IP packet to the client. The client will wait for 10 seconds before sending out another GET_IP request. In the meantime the server checks if all IP addresses are still being used. Therefore it polls each client by sending up to three ICMP echo requests within three seconds.
- A client moves to the *PASSIVE* state once it has got its IP address.
- If a client in the *QUERY* state gets SET_IP responses from more than one server or gets no SET_IP response at all after sending out five GET_IP requests

over five seconds, it broadcasts a REQ_ERR packet and moves to the *NEGOTIATE* state.

- Any client that receives a REQ_ERR packet or a CLIENT_STAT packet will move to the *NEGOTIATE* state. Within this state, all clients broadcast three CLIENT_STAT packets that contain their MAC address, the age of their own copy of the address table and their willingness to run the service. A client that has not received an address table yet uses zero for the age entry. The client with the newest table and the highest availability wins the race and becomes the new server. If there are two or more candidates, the client with the lowest MAC address is chosen. If there are only clients with no local address table, the chosen client will initialise its address table and will assign itself an IP address. The new RHCP server moves to the ACTIVE state and broadcasts a NEW_SERV packet to inform all clients about the decision. Since all clients run the same algorithm, they accept the decision and update their address table if required (the table is part of the NEW_SERV packet payload). The clients move to the *PASSIVE* state or the *QUERY* state if the client has not received an IP address yet. A client that disagrees with the decision can initiate the selection process one more time by broadcasting another REQ_ERR packet.

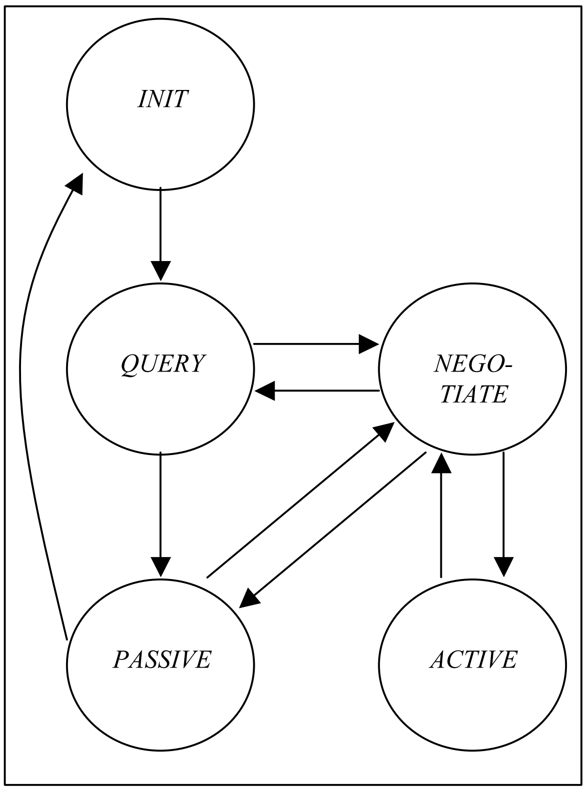

Figure 1: State transitions of a network device.

- Whenever a client moves from the *NEGOTIATE* state to the *PASSIVE* state, it verifies the address table in the NEW_SERV packet. If it is not registered in the table it has to go back to the *INIT* state.
- A client that moves from the *NEGOTIATE* state back to the *QUERY* state resends a GET_IP packet.
- Whenever a client receives a SET_IP packet it checks if its own assigned IP address is properly set in the attached address table. If there is a mismatch the client moves to the QUERY state.

B. Protocol Details

The address table, which is stored by all network clients and which is part of the SET_IP and the NEW_SERV packets, has the following format:

- An 8-bit value *n* to describe the range of IP addresses to be assigned (e.g. 192.168.n / 8). This value is fixed and does not change over the lifetime of an ad-hoc network.
- An 8-bit value *count* describing the number of IP addresses already being assigned.
- A 16-bit age counter that is increased by one, whenever the table changes. The value zero indicates that the table is empty.
- A 255-bit wide bitvector that describes which address in the 192.168.n / 8 range is currently being used. Exactly *count* bits of this vector are set to one.
- An array containing *count* MAC addresses. They are listed in the appropriate order to map them to IP addresses by parsing the bitvector.

A GET_IP packet consists of an 8-bit packet identifier (10001111b) and the (unique) MAC address of the sending client. The MAC address will be copied into the address table of the server to associate an IP address with the MAC address.

A SET_IP packet response by the server consists of

- an 8-bit packet identifier (10001100b),
- the MAC address of the client,
- an 8-bit address suffix *x* that completes the client's IP address to 192.168.n.x,
- an 8-bit address suffix *v* that completes the server's IP address to 192.168.n.v and
- a copy of the address table.

The advantage of this packet format is that both IP address assignment and table updates are done simultaneously. If the server is disconnected while sending out the packet, neither the inquiring client nor the other network devices will receive a message. The client is forced to resend another GET_ID request or a REQ_ERR packet. The dual-server approach as suggested by Cisco in contrast updates the secondary server after an IP address assignment. This results in an address assignment inconsistency, if the server is disabled between these two steps.

A client has to accept any assigned IP address. An additional handshake between client and server (i.e. the exchange of DHCPREQUEST and DHCPACK packets as mentioned in Section II.A. and as specified in RFC 2131) would make a redundant implementation of this service very complex, since a server failure during this handshake had to be taken into account.

The NO_IP packet response consists of an 8-bit packet identifier (10000011b) and the MAC address of the rejected client. Since multiple clients can send IP_GET packets simultaneously, the MAC address is used to identify the receiver of the packet.

The REQ_ERR packet consists of an 8-bit packet identifier (01001111b), the MAC address of the sending client and an 8-bit field describing the reason for the forced service renegotiation. Valid entries for this field are

- MULTIPLE_SERVER_RESPONSES (00001111b),
- NO_SERVER_RESPONSE (00001100b) and
- NEW_SERVER_REJECTION (0001001b).

A CLIENT_STAT packet consists of

- an 8-bit packet identifier (00101111b),
- the MAC address of the client,
- a 16-bit age counter copied from the address table and
- an 8-bit value describing the availability of the client, whereby 0x00 describes highest and 0xFF lowest willingness to become the new RHCP server.

Since CLIENT_STAT packets are broadcasted by all clients, each of them can check individually, if it's own aptitude to become the new service provider is rated higher than the aptitude of it's competitors.

Finally a NEW_SERV packet as send by the new RHCP server consists of

- an 8-bit packet identifier (00011111b),
- an 8-bit address suffix *v* that completes the server's IP address to 192.168.n.v and
- a copy of its address table.

Every client can use he NEW_SERV packet to update its local copy of the address table.

The IP address of the server is not required to run the service, but it provides all clients with an additional piece of information for the status monitoring.

Every client does the status monitoring individually. In it's simplest implementation each client copies every updated address table together with a timestamp into a log file that is stored locally on-board on non-volatile memory. The timestamp is based on each client's system clock.

IV. Results and Future Work

The RHCP protocol is currently being implemented and tested on a Rabbit 2000 architecture with integrated 10 Mbit Ethernet

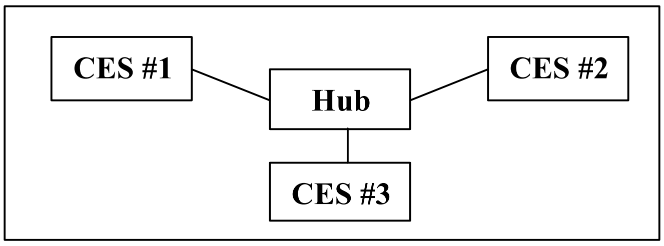

Figure 2: Components of the testbed.

interface. The Rabbit 2000 is an 8-bit microcontroller based on the Z180 processor core with a clock speed of 30 MHz. The IDE consists of a C-compiler, linker, debugger and a number of libraries including a TCP/IP stack and a µC/OS-II real-time operating system kernel [8].

A network with star topology is used as testbed. It consists of three collaborating Rabbit systems and a hub (see Figure 2). Preliminary tests have shown, that the protocol provides the envisaged redundancy in a dynamic ad-hoc network as simulated with the testbed.

The work on this protocol is still ongoing. Future tasks will include

- the seamless integration of a redundant directory service like SLP (Service Location Protocol),
- the definition and implementation of authentication procedures suitable for embedded systems in an ad-hoc network,
- the integration of secure inter-client communication mechanisms like IPSec transport mode SAs and
- the implementation of enhanced monitoring/logging features.

References

[1] R. Droms, "Automated Configuration of TCP/IP with DHCP", IEEE Internet Computing, July-August 1999, pp. 45-53.
[2] Cisco Systems, "DHCP failover", 2000. Available: http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/ciscoasu/nr/nr3.0/concepts/cg03.htm.
[3] K. Kinnear and R. Droms, "Network Working Group Internet Draft – DHCP Failover Protocol", 2003. Available: http://www.ietf.org/ids.by.wg/dhc.html.
[4] S. Cheshire and B. Aboba, "Dynamic Configuration of IPv4 Link-local Addresses", Internet draft, Internet Engineering Task Force, Zeroconf Working Group, March 2001.
[5] S. Thomson and T. Narten, "IPv6 Stateless Address Auto-configuration", RFC 2462, Internet Engineering Task Force, Zeroconf Working Group, December 1998.
[6] S. Nesargi and R. Prakash,, "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network", IEEE Infocom 2002, pp. 1060-1068.
[7] C.E. Perkins, J.T. Malinen, R. Wakikawa, E.M. Belding-Royer and Y. Sun, "IP Address Autoconfiguration for Ad Hoc Networks", Internet Engineering Task Force, MANET Working Group, July 2000.
[8] J.J. Labrosse, "MicroC/OS-II, The Real-Time Kernel", 2nd Edition, CMP Books, 2002, ISBN 1-57820-103-9.