Eindhoven University of Technology

MASTER

Web controllable devices: concept and design

Aalderink, H.A.

Award date:
1999

Disclaimer
This document contains a student thesis (bachelor’s or master’s), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
WEB CONTrollable DEVICES,
CONCEPT AND DESIGN.

ing H.A.Aalderink

Coach: dr.ir.A.C.Verschueren
Coach: ing A.H.J.G. Lommen (TNO Institute of Industrial Technology)
Supervisor: prof.ir.M.P.J.Stevens
Date: August 1999
Abstract

This master's thesis is written as part of the M.Sc. course in information technology at the department of electronic engineering of the Eindhoven University of technology. All work in this thesis is the result of a joined project with TNO (Netherlands Organisation for Applied Scientific Research) Institute of Industrial Technology.

This thesis explores the concept of web controllable devices with respect to:

- Choice of protocol stack and location of functionality: Telnet TCP and IPv6 on the device and HTTP on the server which contains the web pages and CGI-script too.
- Choice of a pipelined data processor implementation of the protocol stack on the device.
- Discussion about Internet delay and the proposal of safety envelopes
- Discussion of modelling the Internet delay in controlled systems.

The design of the chosen pipelined data processor discusses:

- Testability of the designed protocol processor.
- The processing architecture
- A general interface to data-communication circuits
- A bus structure to process the extension headers in the order in which they appear in the IPv6 Payload package.
- An unit for insertion of payload length to solve the causality problem in pipelined encoders for variable length messages

The designed units are modelled in synthesisable VHDL to generate design feedback and functional prototypes using a rapid prototyping board populated with FPGA's developed within TNO.
# Web Controllable Devices: Concept and Design

## Contents

1. **CONTENTS**

2. **INTRODUCTION**

3. **WEB CONTROLLABLE DEVICES**
   3.1. **THE PROTOCOL STACK OF WEB CONTROLLABLE DEVICES**
   3.2. **HARD- OR SOFTWARE STACK IMPLEMENTATION**
   3.3. **CONTROLLABILITY OF WEB CONTROLLED DEVICES**
   3.4. **INTERNET DELAY IN A WEB CONTROLLED SYSTEM IN A STEERING CONFIGURATION**
   3.5. **CONTROL IN A REGULATING CONFIGURATION**
   3.6. **THE DEVELOPMENT PROCEDURE**

4. **DESIGN**
   4.1. **THE NETWORK ABSTRACTION SUB-LAYER AND ITS INTERFACE**
       4.1.1. **The needs of the IP protocol-layer**
       4.1.2. **Controlling a variety of LAN's**
       4.1.3. **A Network Abstraction Sub-layer for Serial Lines using SLIP**
   4.2. **IPv6 ENCODER AND DECODER**
       4.2.1. **The IPv6 protocol**
       4.2.2. **IPv6 encoder**
       4.2.3. **IPv6 Decoder**
   4.3. **EXTENSION HEADERS AND THE X-BUS**
       4.3.1. **Extension Header**
       4.3.2. **The X-bus**
   4.4. **LAST MINUTE LENGTH INSERTION**
   4.5. **INTERNET CONTROL MESSAGE PROTOCOL**
       4.5.1. **ICMPv6**
       4.5.2. **The ICMPv6 design**
   4.6. **CONFIGURATION DATA**

5. **MODELS AND FUNCTIONAL PROTOTYPES**

6. **CONCLUSIONS AND FUTURE WORK**

7. **REFERENCES**
2 Introduction

The Internet is growing rapidly and almost every day new ways of using the Internet emerge. The Internet finds its solid base in the Internet standards. The specifications of the Internet standards are publicly accessible via the Internet. The Internet Engineering Task Force (IETF) does the development of new Internet protocols. The IETF research is done in workgroups. The workgroup publishes its findings in Requests For Comment's (RFC's). These RFC's are not standards, they are just proposals that could end up as internet standard. The latest version of the basic Internet Protocol (IPv6) is currently in this process and has some very desirable properties (like longer address fields) but also some awkward ones (the composition of the message header is no longer fixed, making it difficult to handle these messages in hardware).

The expansion of embedded systems is a less visual, due to its stealthy character. Almost all electronic devices have smart user-interfaces and remote controls. The use of embedded systems has opened the possibility to design user-oriented interfaces rather than letting user interfaces be dictated by the mechanical and electrical demands of a device's function.

The next logical step in the evolution of embedded devices is to connect the embedded system to the Internet. More devices will have Internet connectivity. At first only industrial systems, but in the end even household appliances will have an Internet connection. Imagine programming your VCR at the other side of the globe with an organiser.

The development and concept of web controlled devices contains issues from different scientific fields: Controlled systems, queuing theory, computer science and telecommunication.

In this thesis, we will develop an IPv6 controller which can be integrated with the hardware of an embedded system to form a 'system on a chip' with internet connectivity.
3 Web Controllable Devices

Embedded systems are used in a lot of electrical devices. They are used as controllers for all kinds of systems and as intermediates between a system and a user-interface. So the user interface is not directly connected with the controlled system. This is depicted in figure 3.1

![Diagram of user interface, embedded system, and controlled system]

Figure 3.1 Embedded system as intermediate.

This means the user interface could be placed at another location, as in remote controls. When a user-interface and the embedded system are connected through the Internet, the device could be controlled from every Internet equipped location. The user interface could be integrated in a web page, so the device would be web controlled.

This concept raises some questions:

- Which Internet protocols should be used to connect the web browser to the device?
- Should the protocols in the device be developed in hardware or in software?
- What are the security aspects?
- Which kinds of control could be developed and what are their limitations?
- What are the effects of the packet delay on the controllability?

These issues are discussed in paragraphs.
3.1 The protocol stack of web controllable devices

The web provides a platform independent way to exchange information worldwide. When a device is connected to the web, it would be possible to control it from any web browser on the globe. It would be another level in remote control with enormous possibilities.

A web browser uses HTTP (HyperText Transfer Protocol) to communicate to a HTTP server. The browser requests the web page from this server. The server sends the requested HTML (Hyper-Text Mark-up Language) file to the browser. When the web page contains images or other objects the browser requests them separately when the HTML-file is processed. Interactive web pages contain CGI (Common Gateway Interface) requests. When an HTTP server receives the CGI request it starts the requested program. The output of the program is redirected by the server to the browser.

The web page of the web controlled device must be stored somewhere. When the page is stored within the web-controlled device, every device needs a storage facility for its own page. The page could also be stored on a server, generally this will be a local server. This opens the possibility to create several web pages for one device. These pages can contain user tailored interfaces on a need-to-control (as in need-to-know) base. Bearing in mind a local server is generally present, user tailored interfaces and the costs of the storage medium, storing the interface web page on the server is the best choice.

To access the web-controlled device the web browser uses HTTP to request the web page. The HTTP server sends the web page to the browser. Since the page is on a server, it is obvious to place the HTTP server functionality on that server. This makes that server a web server. Generally the local web server is chosen to perform this task.

Unfortunately there exists no Internet standard protocol to control a device directly from a web page. So the device has to use another protocol to interface between the web browser and the device. The most obvious choice would be to use the TELNET protocol to control the device. This protocol can be used to create a textual command interface. So the device would not only be web-controllable, it will be net-controllable. Since the World Wide Web is one of the Internet applications, the net controllable device would have more possibilities than the web controllable devices. It could be controlled by a web browser or a telnet application and would be able to send and receive e-mail. On top of that it could be controlled by another device.

Additional software is needed to generate the textual commands in response of the interactive web components (e.g. buttons, dialog boxes, etc). The interactive web components need handlers. The handler function can be performed by an applet running on top of the browser or a cgi-script or servlet running on a server. This generally would be the server that contains the web page. These two possibilities are depicted in figure 3.2
Both options result in the same requirements for the device. The selection of an option is mainly a security-related subject. When the handler is situated on a server behind a firewall, the firewall could block direct access to the device. This way the device can be protected against unauthorised actions via the net. When the handler is located on the client, hostile hosts can imitate the client and access the device.

The choice for a separate HTTP server over an embedded one results in a net controllable device. The Telnet protocol is used to control the device with a textual command language. Since the Telnet protocol runs on top of the TCP protocol the device should have a protocol stack as depicted in figure 3.3

<table>
<thead>
<tr>
<th>Textual controllable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telnet</td>
</tr>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>IP</td>
</tr>
<tr>
<td>LAN or Point to Point</td>
</tr>
</tbody>
</table>

Figure 3.2 Distributions of protocols for web controllable devices

Figure 3.3 The protocol stack of web controlled devices
Web controllable devices, concept and design.

The native communication medium, a LAN or a Point to Point connection is at the bottom of the stack. It connects the device to the Internet. The Internet Protocol (IP) uses the native medium to provide a best effort packet delivery service to the Transport Control Protocol (TCP) layer. The Transport Control Protocol delivers a byte stream to a certain application at a certain host. The Telnet application provides a terminal interface that uses TCP to communicate.

At the moment of writing there are two versions of the Internet Protocol (IP), version 4 and version 6. The development of IPv6 started because IPv4 had run out of IP-addresses. Currently both are used within the Internet community. The development of a new version of IP opened the opportunity to get rid of some imperfections of version 4. The difference between version 4 and version 6 concerns the areas of:

- Expanded Addressing Capabilities
  
  IPv6 increases the IP address size from 32 bits to 128 bits, to support more levels of addressing hierarchy, a much greater number of addressable nodes, and simpler auto-configuration of addresses. The scalability of multicast routing is improved by adding a "scope" field to multicast addresses. And a new type of address called an "anycast address" is defined, used to send a packet to any one of a group of nodes.

- Header Format Simplification
  
  Some IPv4 header fields have been dropped or made optional, to reduce the common-case processing cost of packet handling and to limit the bandwidth cost of the IPv6 header.

- Improved Support for Extensions and Options
  
  Changes in the way IP header options are encoded allows for more efficient forwarding, less stringent limits on the length of options, and greater flexibility for introducing new options in the future.

- Flow Labeling Capability
  
  A new capability is added to enable the labeling of packets belonging to particular traffic "flows" for which the sender requests special handling, such as non-default quality of service or "real-time" service.

- Authentication and Privacy Capabilities
  
  Extensions to support authentication, data integrity, and (optional) data confidentiality are specified for IPv6.

At the moment of writing IPv6 is not an Internet standard, but this is merely a question of time. Currently it has the status of draft standard. This means it has at least two independent
interoperable implementations and some operational experience has been obtained. It will become a standard when the protocol has proven to be well behaved on a variety of platforms and when sufficient operational experience has been gained.

Clearly version 4 and 6 of IP are not compatible. So the Internet Engineering Task Force (IETF) developed a transition scheme that allows a gradual introduction of version 6. First IPv6 is used on top of IPv4, creating IPv6 pools within the IPv4 environment. These pools will grow and finally interconnect. Within the IPv6 pools IPv4 is tunneled through IPv6. This will result in shrinking IPv4 pools within the IPv6 environment. Finally when the IPv4 pools have disappeared the IPv4 protocol can be removed as a standard.

IP version 6 should be chosen for the web controllable devices. Although version 6 has the status of draft standard it is not likely to change significantly. The privacy and flow label developments should be supported within the implementation of the web controllable devices, since they would be able to provide real time control and user authentication.

3.2 Hard- or software stack implementation.

The protocol stack for web controlled devices can be implemented in hard- or software. A key factor in the decision is the system load caused by the implementation. Since the main task of the embedded system is to control the system, the protocol stack implementation should not be too large load on the embedded system.

Interprophet has developed an Ethernet card with TCP on board (SiliconTCP). Tests performed by Interprophet [Interprophet] of the bandwidth of their SiliconTCP (hardware) implementation showed, that their silicon implementation is capable of sending 9MBps while the WinNT (software) implementation could do 5MBps. This resulted in a CPU load of 2% with the Silicon TCP implementation and a 98% receiver and 48% transmitter load with the WinNT stack. This is mainly the result of the added dedicated hardware. It would be interesting to compare a card with a dedicated CPU and the SiliconTCP implementation in a similar test.

From the tests performed by Interprophet could be concluded that a web controllable device with dedicated hardware is preferable. The protocol stack generally has the structure of a data processing pipeline. This structure can be implemented in hardware with a high degree of parallelism and would therefore be able to achieve higher speeds.

3.3 Controllability of web controlled devices.

The control of a device through the Internet can be done with two conceptually different methods. In the first method the Internet is used to control the device in a steering configuration, in the second the Internet controls the device in a regulating configuration.
Web controllable devices, concept and design.

The general controlled system (figure 3.4 A) has a preferred value as an input and the actual value of the controlled quantity as an output. The controller (C) uses the difference between the actual and preferred value to calculate its output to correct the controlled process (P).

The web controlled system in a steering configuration (figure 3.4 B) gets its preferred value through the Internet. The output of the process is sent back through the net to enable monitoring of the controlled process output. The effect of Internet delay in this system is discussed in section 3.4.

In a web controlled system in a regulating configuration (figure 3.4 C) the controller calculates the difference between the preferred value and the output that is relayed back through the Internet. The process input is sent through the Internet to the controlled process. The effect of Internet delay in this system is discussed in section 3.5.

3.4 Internet delay in a web controlled system in a steering configuration.

The Internet delays the information send through the Internet. Therefore the web-controlled system will not respond directly to the steering information and on top of that the browser will show delayed system status information. This raises questions about safety of web controlled devices. When an action is performed and the situation becomes dangerous, a correcting action could arrive to late and a fatal situation occurs. To prevent this a safety envelope is defined. How this could be done is shown by an interactive model railroad.

The interactive model railroad [Rail Road] has 2 trains (A and B) and 3 stations (1,2,3) as depicted in figure 3.5.
Web controllable devices, concept and design.

Figure 3.5 Model railroad with three stations.

Setting the speeds of both trains and controlling the points of the track controls the model railroad locally. In a situation of a potential collision, taking appropriate actions prevents the collision. When the same interface is used from a remote location and the status display shows a potential collision, and the appropriate actions are taken, several things can happen:

- Since the status information is delayed the trains could have collided before the dangerous situation could be noticed.
- Since the correcting action is delayed, the trains could have collided before the correcting action is applied.
- When the correcting action arrives in time the collision is prevented.

When the remote interface is designed to send the system from one stable situation to another stable situation by a safe transition, dangerous situations don't occur. This model railroad has three stations. Since trains tend to stop at stations, a train at a station will be regarded a stable situation. Since there are three stations and two trains there are nine stable situations in this railroad model. Since two trains at a station will cause the possibility of a collision, two trains at a station will be regarded as unsafe. So the model has nine stable situations. These stable situations contain six safe situations and three unsafe situations. In this model the control space consists of all reachable situations. Since the controls of the remote interface are restricted to directing trains to stations, the control space contains all combinations of trains and stations. The control space for this model is depicted in table 3.1

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train B</td>
<td>X</td>
<td>&amp;</td>
<td>&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 3.1 Control space

Locations with a '&' are safe and locations with a 'X' are unsafe. Since only safe situations are allowed the safe situations are in the safety envelope.

Imagine train A at station 2, train B at station 1 and the trains have to swap places. In other words a route through the control space from (A=2, B=1) has to be found. The shortest routes are via (1, 1) or (2, 2) but these situations are unsafe because the trains could collide at the station. This means one train has to move to station 3, the second train has to move to its destination and then the first one has to move to its destination. The safe routes are therefore {(2, 1), (2, 3), (1, 3), (1, 2)} and {(2, 1), (3, 1), (3, 2), (1, 2)}. This is depicted in table 3.2.
The table shows one of the unsafe routes (dashed line) and the two safe routes (bold lines). So in general to insure safety a safety envelope is needed. The safety envelope constrains situations in the control space that are both safe and stable. The safety envelope is therefore a subset of all safe and all stable situations, in the control space. Transactions are only be allowed between points within the safety envelope.

The safety envelope and the safe and stable points both can be described as bodies within the control space. In this description every safety envelope body is contained within a safe and stable body and a transition between two safety envelope bodies that are not interconnected is dangerous and thus not allowed. These transactions would only allowed from the controls at the location itself.

So to ensure the safety of the system, a safety envelope within the control space is necessary. The safety envelope ensures that the system is always in a safe situation. The use of a safety envelope may result in safe situations that become unreachable from the remote controls. These unreachable safe situations are the cost of the trade of between safety and control possibilities.

3.5 Control in a regulating configuration

The web-controlled device that is controlled in a regulating configuration has the Internet delay two times in its control loop. (Figure 3.6) To design this type of controlled systems the Internet delay has to be known. Since the route between the device and the web browser could be different form message to message and goes trough a variety of systems, it is difficult to derive a model from the route and used technologies. The Internet delay model can be derived from queuing theory. When the M/M/1 queuing model is applied a Erlang-k distribution (formula 3.1) is found for the total processing time.

\[
P = \frac{\left(\frac{k}{T}\right)^k}{(k-1)!} \cdot e^{-\frac{k}{T}}
\]

Formula 3.1 The Erlang-k distribution.

The formula describes the probability P of the total system time t as a function of t, the (average) number of occupied queue positions k and the average system time T. To get some insight in these parameters for Internet delay some measurements were performed. From a host at TNO(Eindhoven, the Netherlands) the delay to a near host (vwwwww.stack.nl located in Eindhoven) and a remote host (slashdot.org located on the East Coast, U.S.A.) were measured. These measurements were done by logging the result of 10000 ping requests to that host. A frequency distribution plot has been made of these measurements.
The graph shows packages that made the turn around trip within the timeout of 2000ms. The plotted distributions are both Erlang-k distributions, as was expected from the chosen queuing theory model. The parameters of the Erlang-k distribution can only be found with curve fitting techniques, because the queue occupation of the M/M/1 model can not be found with this experiment and the average in the distribution function and the measurements are not the same due to the timeout.

The Erlang-k distribution can be used to model the Internet delay in the control model of the system. The usability of the delay probability function for web controlled systems in a regulating configuration must be investigated.

From the graphs it becomes clear that the system response time can be enhanced when the time out is set just after the peak in the probability distribution. Since the majority of the packets in the graph will arrive within this time-out, lost packages or slow packages are retransmitted swiftly. This will reduce the average delay at the cost of an increase in Internet traffic. When the system applies the data a fixed period after transmission the model becomes a general delay function. When the data has not arrived when it should be applied, the system has to have an extrapolation function to fill the data gap. The controlled system must be robust enough to handle the extrapolated data.

To ensure the route of packets trough the Internet, the use of virtual circuits is recommended. The virtual circuit could provide a guaranteed average bandwidth over a fixed route. These virtual circuits will be profiled by the IPv6 traffic class extension. Since the development of traffic classes is not completed, the investigation of traffic classes has to wait until the development has resulted in a request for comments (RFC). When the virtual circuit for real-time purposes
only uses connection oriented media, a fixed bandwidth over a fixed route will be guaranteed. When the data is encoded with a self-correcting code, the need for retransmissions caused by distortions could be reduced at the cost of sending additional data. The need for these extensions should be investigated further and may possibly result in a proposal for an additional traffic class.

3.6 The development procedure.

The development done within this master's thesis is constrained by a time limit. Since this project is too big to complete within the given time, choices have to be made. These choices must be made bearing in mind the fact that others must be able to finish the project.

Within this thesis only the IPv6 layer and the interface with the native network will be designed. The TCP and telnet layer as well as the interface to the application will not be designed within this thesis.

The designed parts will be tested using a rapid prototyping system developed by TNO Industry. This system allows the description of hardware with a synthesisable hardware description language. This description is then synthesised, routed and fitted on to a programmable logic device. This results in a fully functional prototype of the designed hardware.
4 Design

The web controllable device must have a TCP/IP stack to communicate with the Internet. The structure of a protocol stack encourages the design of a pipelined data processing structure. The pipelined data processing structure consists of cascaded entities. The communication between the entities should be designed bearing in mind the testability of the design. The communication between the entities has to be able to operate in four configurations to be testable. These configurations are depicted in figure 4.1.

![Diagram](image)

Figure 4.1 Testability configurations.

When the inter-entity communication can be used in these four modes testability of the design is guaranteed. The designed data stream-processing architecture is depicted in figure 4.2.
Web controllable devices, concept and design.

Network Abstraction Sub-layer

Figure 4.2 Designed data stream processing architecture.

The "LAN / Point to point connection" is the local medium for the Internet communication. Since the Internet is designed to perform regardless of the media technology, this stack must be able to perform on top of different media technologies as well. This requires the definition of a general interface to abstract from media technologies, the "Network Abstraction Sub-layer Interface" (NASI). The "Network Abstraction Sub-layer" (NAS) contains all media dependent aspects of the Internet connection. The NAS and NASI are discussed in paragraph 4.1.

The "Last Minute Length Insertion" takes care of administrating the length of a packet and includes this information in the packet just before transmission. The Last Minute Length Insertion is discussed in paragraph 4.1.

ICMPv6 is the "Internet Control Message Protocol version 6". The ICMP messages are used for error and status messages. In response the host makes adjustments to the generated traffic or generates additional ICMP messages. ICMPv6 is discussed in paragraph 4.4.
Web controllable devices, concept and design.

The IPv6 Encoder produces the IPv6 packets. The IPv6 Decoder receives the IPv6 packets and filters out all wrong addressed or corrupted packages. This is discussed in paragraph 4.1. Both the IPv6 Encoder and decoder are connected to the X-bus. The X-bus is designed to insert dynamically extension header handlers into the pipeline. The X-bus and the extension header handlers are discussed in paragraph 4.2.

The TCP module contains the Internet transport protocol. The Telnet protocol provides a character based terminal interface to the net controllable application. The Telnet and the TCP module are not discussed in this thesis. The design of these units will be future work.
4.1 The Network abstraction Sub-layer and its interface

The Network Abstraction Sub-layer Interface (NASI) provides to the IP protocol layer a general interface to a variety of communication media circuits. Hence the NASI has to take care of the needs of the IP protocol layer and must be able to control a variety of communication media circuits. The NASI will be specified after a look into the needs of the IP layer and the various ways to control a variety of media circuits.

4.1.1 The needs of the IP protocol-layer.

The IP protocol layer uses a data-communication connection to transport the payload data to its final destination. In order to do so the IP layer needs knowledge about the services of the used medium.

The IP layer has to know the maximum payload size of data for the used communication protocol (prescribed by the protocol standard). For a given protocol the maximum payload size is a constant.

When a LAN is used to transport the IPv6 package to the next host, the LAN needs to know the LAN address of the next host. IP only knows the IP address of the next host. This requires a procedure to find the LAN address for a given IP address. IPv4 uses the Address Resolution Protocol (ARP) to find the network address for an IP host. IPv6 has a 128 bits addressing scheme. This makes direct mapping of network addresses to IP addresses possible. A part of the IP address contains the LAN address. The remaining part of the address is used to differentiate between LAN address spaces. The direct mapping scheme can be found in the IPv6 addressing architecture [RFC 2373]

4.1.2 Controlling a variety of LANs

The Network Abstraction Sub-layer (NAS) has to be able to generate the signals needed to control a variety of LAN circuits. Most LAN circuits are designed to be controlled by microprocessors or a microcontroller. Microprocessors and microcontrollers generally use three types of interfacing:

- Polling
- Interrupts
- DMA

All these interfaces are based on I/O handshake. Hence the Network Abstraction Sub-layer Interface should use a similar handshake that can be used with these devices.

When the NASI provides the information needed for these three methods practically every LAN controller can be used to transport the IP messages. These handshake signals can be used to create the signals used in these methods by a state machine. The width of the data buses is set to 8 bits because the TCP/IP suite is byte oriented.
The Send and Receive stream use the same mechanism. When the source has a message that must be transported, it makes the message signal high. The Request signal is made high by the destination indicating it is ready to receive a byte. The source provides the data and Acknowledge is made high indicating the data is stable and available. When the destination has stored the data, Request is made low. Acknowledge is made low by the source. The destination makes the request signal high again to indicate it wants to receive the next byte. When the last byte is stored the message signal is made low, indicating the end of the message. The destination then makes a possibly pending request low.

This scheme transports 8 bits in 4 CLK ticks. This means a transport rate of $2f_{\text{clk}}$ bits per second.
This interface can easily be connected to a test data source or sink. When Message, Request and Acknowledge are high the data can be copied for listening test purposes. Hence this interface meets the requirements for testability and should therefore be used to interconnect all entities in the design.

4.1.3 A Network Abstraction Sub-layer for Serial Lines using SLIP

For testing purposes an Internet connection on basis of the SLIP protocol is developed. The Serial Line Internet Protocol (SLIP) [RFC 1055] makes it possible to use IP over serial lines. SLIP encodes IP outgoing messages in a byte stream and decodes the incoming byte stream. The encoder and decoder are connected to the NASI and a UART. The encoder sends a frame separation character (0xC0) at the start and end of each message. When the frame separation character appears in the message it is stuffed. The escape character (0xDB) is also stuffed. They are respectively replaced by the (0xDB, 0xDC) and (0xDB, 0xDD) sequences. The UART will operate in interrupt mode and will therefore request the next byte.
4.2 IPv6 Encoder and Decoder

The IPv6 Encoder and Decoder must generate/process messages conform to the IPv6 protocol. The design of the encoder and decoder are discussed after the protocol specification.

4.2.1 The IPv6 protocol

The IPv6 packet contains a header-section and a payload-section. The payload section contains the transported data and extension headers. Extension headers are used to describe the optional features of the IPv6 protocol and are discussed in paragraph 4.3.1. The 40-octet (byte) IPv6 header format is depicted in fig 4.5.

<table>
<thead>
<tr>
<th>3</th>
<th>11</th>
<th>15</th>
<th>23</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Version</td>
<td>Traffic Class</td>
<td>Flow label</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Payload Length</td>
<td>Next Header</td>
<td>Hop Limit</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Destination Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.5 IP version 6 header format.

Version 4-bit Internet Protocol version number = 6. Messages from other versions must not be accepted.

Traffic Class The 8-bit Traffic Class field in the IPv6 header is available for use by originating nodes and/or forwarding routers to identify and distinguish between different classes or priorities of IPv6 packets. At the point in time at which the IPv6 specification is being written, there are a number of experiments underway in the use of the IPv4 Type of Service and/or Precedence bits to provide various forms of "differentiated service" for IP packets, other than through the use of explicit flow set-up. The Traffic Class field in the IPv6 header is intended to allow similar functionality to be supported in IPv6. The IPv6 interface must provide means for an upper layer to supply a value. The default value is zero. The upper layer must not assume that the received value is equal to the sent value.

Flow Label 20-bit flow label. The 20-bit Flow Label field in the IPv6 header may be used by a source to label sequences of packets for which it requests special handling by the IPv6 routers, such as non-default quality of service or "real-time" service. This aspect of IPv6 is, at the time of writing, still experimental and subject to change as the requirements for flow support in the Internet become clearer. A host without flow label support must send a zero flow label and must ignore the received flow label.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload Length</td>
<td>16-bit unsigned integer. Length of the IPv6 payload, i.e., the rest of the packet following this IPv6 header, in octets. (Note that any extension headers present are considered part of the payload, i.e., included in the length count.)</td>
</tr>
<tr>
<td>Next Header</td>
<td>The Next Header field is an 8-bit selector. It identifies the type of header immediately following the IPv6 header and uses the same values as the IPv4 Protocol field [RFC-1700 et seq.].</td>
</tr>
<tr>
<td>Hop Limit</td>
<td>8-bit unsigned integer. Decrement by 1 by each node that forwards the packet. The packet is discarded if the Hop Limit is decrement to zero.</td>
</tr>
<tr>
<td>Source Address</td>
<td>128-bit address of the originator of the packet.</td>
</tr>
<tr>
<td>Destination Address</td>
<td>128-bit address of the intended recipient of the packet (possibly not the ultimate recipient, if a Routing header is present).</td>
</tr>
</tbody>
</table>
4.2.2 IPv6 Encoder

The header encoder generates the IPv6 package from the incoming byte stream from the X-bus (as depicted in figure 4.2). The packet related header fields are encoded as in-stream data leading the payload. The format of the incoming payload is depicted in figure 4.6.

![IPv6 encoder incoming byte stream.](image)

The Next Header field occurs twice. The first instance is used in the header. The second is used to select an extension header encoder, this is discussed in detail in paragraph 4.3.2. The header encoder produces an IPv6 package (white area in figure 4.7) with the length field set to zero and the required last-minute-length-substitution commands (gray area). This is discussed in paragraph 4.4. The output of the IPv6 encoder is sent to the last minute length insertion unit.

![IPv6 encoder output stream.](image)

The Length location command indicates the location of the length field. This field will end up as 16 bits payload length. The Store Length command informs the last minute length insertion unit to insert the length value, that follows the store command, at the indicated location.
4.2.3 IPv6 Decoder

The header decoder receives the byte stream. It has to:

- check the protocol version
- check the IP destination address and pass the source address to the next level.
- check the packet length
- start the extension header decoder

As a result of these checks a packet can be passed on or discarded. When a packet is discarded all receiving modules are signalled to flush the message out of the processing pipeline. In case of an error the ICMPv6 is signalled and sends the appropriate ICMP message. The format of the decoder output stream is depicted in figure 4.8.

\[
\begin{array}{|c|}
\hline
\text{Payload Length} \\
\hline
\text{Source Address} \\
\hline
\text{Destination Address} \\
\hline
\text{Next Header} \\
\hline
\text{Payload} \\
\hline
\end{array}
\]

Figure 4.8 IPv6 decoder output stream.

The payload length includes the space used by the extension header. Since the TCP header does not contain a payload length [Comer], it has to calculate the payload length from the IPv6 payload length and the length of the extension headers and the TCP header.

Since the TCP layer uses IP addresses to identify a connection, the decoder output contains the source and destination addresses.

The next header field contains an identifier of an extension header or an upper layer protocol. This field is used to address the appropriate handler.
4.3 Extension headers and the X-bus.

The IPv6 header contains a next header field. This field is used for upper layer protocols and optional IPv6 extensions. The use of the next header field and the extension headers are discussed in 4.3.1. The X-bus is designed to select the appropriate extension handler for an extension header. The X-bus is discussed in 4.3.2.

4.3.1 Extension Header

The next header field identifies the upper layer protocol and extension headers. These extension headers are a part of the payload and contain optional services. The extension header also contains a next header field. The payload contains a linked list of extension headers (figure 4.9). At the tail of this linked list is the upper layer protocol.

<table>
<thead>
<tr>
<th>IPv6 header [next header = TCP]</th>
<th>TCP header + data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6 header [next header = Routing]</td>
<td>Routing Header [next header = TCP]</td>
</tr>
</tbody>
</table>

Figure 4.9 Linked list of protocol headers.

The currently defined extension headers are:

- Hop-by-Hop Options header
- Destination Options header (note 1)
- Routing header
- Fragment header
- Authentication header (note 2)
- Encapsulating Security Payload header (note 2)
- Destination Options header (note 3)

Note 1: for options to be processed by the first destination that appears in the IPv6 Destination Address field plus subsequent destinations listed in the Routing header.

Note 2: additional recommendations regarding the relative order of the Authentication and Encapsulating Security Payload headers are given in [RFC-2406].

Note 3: for options to be processed only by the final destination of the packet.

A host must encode extension headers in this order. When extension headers are decoded any order of extension headers should be processed. At the first glance these requirements seem strange, but these requirements allow the definition of a second order of extension headers without the need for updating all implementations of IPv6. The receiver of the message has to route the message through the appropriate extension handlers to meet these requirements. The X-bus is designed to perform the dynamic concatenation of extension handlers. In the Transmitter stack the X-bus is applied as well. The X-bus is applied in the transmitter stack to insure that additional orders of extension headers can be implemented and to ease the testing of extension header encoders and decoders.
4.3.2 The X-bus.

The X-bus is designed to transport a message through all extension handlers needed to process the extension headers. These extension handlers have to form a dynamic pipeline in order to process the headers in the received order. The X-bus is time multiplexed to transport data to and from all extension handlers. This is shown by the case depicted in figure 4.10.

The topology of the X-bus in this case is depicted on the left-hand side. The X-bus is connected to the data source IP and the destination TCP, the bus controller and the extension handlers A, B, C and D. When the message, displayed in the top right corner, is received by IP, IP processes the message. The output of IP is placed on the bus when the controller indicates time slot zero. The data in time slot zero is read by TCP as depicted in the bottom right corner. Since IP knows the fields in the IP-header, IP knows when the next header field is placed on the bus. At that moment IP signals all extension handlers a next header field is on the bus. Handler C recognises its next header value and signals this to the controller. The controller assigns the input of handler C slot zero and the output of C slot one. The TCP input is assigned slot one also. Now all data from IP is processed by C before TCP processes it. When C places the next header field of its header on the bus it signals all the remaining handlers. Now A recognises its next header value and signals the controller. The controller assigns the output slot of C to the input of A and the output of A to the input of TCP. So A is placed between C and TCP in the pipeline. When A encounters its next header field it signals the handlers too and B responds. The controller places B between A and TCP. The next header field of B contains the TCP next header value. TCP recognises its next header value and knows the next data is its header. At the end of the message all handlers signal the end of the message and the controller returns the situation to its original state.
Web controllable devices, concept and design.

When the controller cyclically calls and assigns all slot numbers in increasing order a number of straightforward processing rules can be derived:

- Every handler only needs to know its protocol to signal other handlers.
- IP always places its data in slot 0
- TCP always gets its data from the slot with the highest number.
- A handler puts its output data always in the slot after its input.
- A handler gets its input from the slot that contained its header-id when it was signalled.

On top of the dynamic pipeline a handshake is needed to control the communication between handlers. The handshake as proposed for the NASI (section 4.1.2) will suffice, because it leaves room for pipeline bubbles to travel in both directions. The complete X-bus contains:

- X-Slot that indicates the active slot.
- X-Data that contains the transferred data.
- X-Request, the data request signal.
- X-Acknowledge, the acknowledge signal for the data.
- H-Request the request signal for building the dynamic pipeline.
- H-Acknowledge the acknowledge signal for building the pipeline.

The data transfer handshake signals are depicted in figure 4.11.

![data transfer handshake](image)

Figure 4.11 data transfer handshake signals

When X-Slot indicates the time slot of the connection between the transmitter and receiver, the transmitter makes X-Message high when it has a (remainder of a) message to send and the receiver indicates with X-Request when it wants to receive data. When both X-Request and X-Message become high in response to the time slot, the transmitter places the data on X-Data and indicates it contains valid data with X-Acknowledge.
The pipeline setup handshake is depicted in figure 4.12. The transmitter is IP or an extension handler and the receiver is generally TCP or another high level protocol. The controller is the bus controller and the handler is the handler of the next header. When the transmitter wants to place the next header field on the bus it indicates this with H-Request. When the data is transferred between the transmitter and receiver this is indicated by X-Acknowledge as described previously. Normally the handler stores every value of X-Slot. When H-Request is raised the handler does not store the slot value but looks at X-Data and X-Acknowledge. When the valid data contains the next header value of the handler, the handler conforms it has recognised its header. Now the handler will take data from the stored slot number and places its output in the next slot.

The mapping of the signals that belong to one connection relative to the slot that contains the time slot for this connection is depicted in figure 4.13. The distribution of the signals has been chosen to match the timing of the network abstraction sub-layer interface. When less than five handlers are used it takes four clock ticks to process one byte. When more than four handlers are necessary the processing takes the number of clock ticks per byte equal to the number of headers. When a handler that does not perform any action on or with the payload would be able to get out of the pipeline the processing speed would increase. This means the counter has to skip the input slot numbers of the disconnected handlers. This makes the controller more complex. On top of that the output of IP will not be always in slot 0. When the handler that gets
its data from slot 0 disconnects and IP continues to place its data in slot 0 the data is never passed on. Additional research is needed to develop a scheme that allows a handler to disconnect.

The X-bus is used at the transmitter side for two reasons. The X-bus on the transmitter side keeps the possibility for other extension header orders open and it eases the testing of an encoder decoder pair for a extension. For testing purposes they can simply be placed on one bus.
4.4 Last minute length insertion.

An IPv6 message can contain several headers:

- The IPv6 header
- Optional extension headers
- A higher level protocol header

Since the payload follows the IPv6 header in the pipeline a causality problem occurs. The length of the payload has to be known when the IPv6 header is constructed, but the length of the payload can be determined after the payload has passed through the IPv6 encoder. This problem occurs in all pipelined variable length package encoders. There are three basic solutions.

- Fix the length of the package and include the real length at the end of the package. Since this is not according to the IPv6 specification, the solution is discarded.
- When the length of all the extensions and the upper layer message is known in advance the length of the payload could be calculated in the header encoder. Since the extension can perform a transformation on the upper layer data, the length of the package is depending on the used extensions and the payload data. So this option would restrict the transformations on the data to fixed length transformations. Since this is not conform the IPv6 extension header specification this solution can not be applied.
- The package has to be delayed in order for the length information to catch up with it. This cannot be done with a simple finite state machine, but requires at least a push down machine.

The last minute length insertion is a solution of the last category. The last minute length insertion solution stores the complete package. The protocol encoders use in-stream signalling to inform the last minute length insertion. Since the protocol encoder knows how the length of the package has to be determined the encoder counts the package length. Since the protocol encoders don't have to store the package, they can be final state machines. Since the package is stored only ones, the amount of needed memory resources is reduced and processing is delayed only ones. When the last minute length encoder can constrain two packages at the same time one packet can be sent while the other packet is constructed.

The in-stream last minute length insertion commands can be easily embedded in the data stream when the data width of the stream is changed to 9 bits. The data and commands are mapped onto the 9 bit as depicted in table 4.1.

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 [Data]</td>
<td>Data</td>
</tr>
<tr>
<td>10 [Level], 1[High length byte], 1[Low length byte]</td>
<td>Length value command followed by two bytes of data</td>
</tr>
<tr>
<td>11 [Level]</td>
<td>Length location command.</td>
</tr>
</tbody>
</table>

Table 4.1 Last minute length insertion commands.

The level value differentiates between length values (up to 128 of them). The lead zero of the data indicates the contents is data and makes it easy for the encoders to cope with the last minute length insertion commands. When there is no leading zero the data has to be passed on
unchanged. Since the location of the length value will end up as two bytes, the counters of the encoders have to take this in account. So when the bit pattern has two leading ones the counter in an encoder has to be increased by two.

The last minute length insertion unit will be quite basic. The unit must have an address counter to indicate the place where the packet data has to be stored in memory. When a length location command arrives the counter value is stored in a table at the row indicated by the level value. When the Length value command arrives, the packet storage memory will be addressed with the value at the row indicated by the level value and the length information will be stored in the packet buffer.

4.5 Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) is used to inform hosts about errors and lost packages. A new version of ICMP is developed by the IETF. This version is adapted to IPv6 and is therefore called ICMPv6 and is described in RFC 2463.

4.5.1 ICMPv6

An ICMPv6 message is sent as a payload in an IPv6 message and is identified by a next header value of 58. A general ICMPv6 message format is depicted in figure 4.15.

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The type field describes the type of message. The lowest 128 types are error messages and the highest 128 types are informational messages. The types defined in (RFC 2463) are shown in table 4.2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Destination Unreachable</td>
</tr>
<tr>
<td>2</td>
<td>Packet Too Big</td>
</tr>
<tr>
<td>3</td>
<td>Time Exceeded</td>
</tr>
<tr>
<td>4</td>
<td>Parameter Problem</td>
</tr>
<tr>
<td>128</td>
<td>Echo Request</td>
</tr>
<tr>
<td>129</td>
<td>Echo Reply</td>
</tr>
</tbody>
</table>

The code field provides an extra level of message granularity and is therefore dependent of the message type. The checksum field is used to detect corrupted messages.

The content of the message body is completely dependent of the message type. A summarisation of message body contents and the type of messages, in which they occur, are placed in table 4.3. The detailed description of the messages can be found in RFC 2463.
Web controllable devices, concept and design.

<table>
<thead>
<tr>
<th></th>
<th>Destination Unreachable</th>
<th>Packet Too Big</th>
<th>Time Exceeded</th>
<th>Parameter Problem</th>
<th>Echo Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>As much of the invoking packet as will fit without exceeding the minimum IPv6 MTU (= 1280 octets) (MTU = Maximum Transfer Unit)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>MTU</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Pointer to erroneous field</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X X</td>
</tr>
<tr>
<td>Sequence Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X X</td>
</tr>
</tbody>
</table>

Table 4.3 Body contents of ICMPv6 messages

When an event causes a host to send an ICMPv6 message, it generally sends the message to the source of the packet that caused the event. This address could be a special purpose address, like anycast or multicast addresses. The host could also have several IP addresses. The ICMPv6 specification contains a set of rules for these situations.

4.5.2 The ICMPv6 design

The shape of the ICMPv6 unit in figure 4.2 looks strange at first sight. The unit is connected to the X-bus in the receiving pipeline, the X-bus in the transmitting pipeline and between the NAS and IPv6.

Since the ICMPv6 messages are sent as if they are originating from higher level protocol, the transmitter must be in the transmitter pipeline as a high level protocol. The ICMPv6 transmitter is triggered by events generated by the surrounding units to generate the ICMPv6 message that is related to the event. In order to generate a body that contains the invoking packet or a pointer to the erroneous field the ICMPv6 unit has to collect information about the processing state of the package.

The ICMPv6 unit resides between the NAS and IPv6 Decoder to store the header and the extension headers of the message. When an error occurs in the header or one of the extension headers the ICMPv6 unit uses the stored headers to regenerate the processed part of the message to include the invoking package in the ICMPv6 message. After regenerating the processed part, the unprocessed part is re-routed from the NAS to the ICMPv6 transmitter. Since the storage unit needs a memory address counter to store the data, this counter can also be used to generate the pointer to the erroneous part of the message.

The ICMPv6 unit also resides on the receiver X-bus. Received ICMP messages must be handled either within the stack or by the by the embedded system. This needs further study of the actions that have to be taken in response to a message.

The ICMPv6 message decoder must handle the received ICMPv6 messages and generate error messages to the application and error events to the handlers. The communication between the ICMPv6 unit and the header encoders and decoders need more study.
4.6 Configuration data

The TCP/IPv6 protocol stack needs a number of configuration parameters (e.g. IPv6-address of the web controllable device, Encryption end authorisation keys, Time to live value, etc). These configuration parameters are stored in the Configuration data unit. The unit has an interface that allows adjustments of the configuration. Generally this unit will have the character of a memory device. The internal structure of this unit and the communication with other units are subjects for future investigations.
5 Models and functional prototypes

During the design process models were developed to produce feedback on the design process. The models are described in synthesiseable VHDL with Summit Visual HDL. The top level of the model is depicted in figure 5.1.

Figure 5.1 Top level model
Web controllable devices, concept and design.

The top layer of the VHDL description contains the units of the protocol stack. The NAS_SLIP unit has a serial interface to the Internet (SerialIn and SerialOut). The NAS_SLIP unit is connected to the IPv6 unit, the signals between the NAS_SLIP and the IPv6 unit form the Network Abstraction Sub-layer Interface as described in section 4.1.2. The TCP unit contains test modules. This module generates a message to be sent and absorbs a received message. The top layer model is used to test the interaction between modelled units. Within each test a case is used that follows a specific path through the design.

![Diagram of NAS_SLIP unit](image)

Figure 5.2 Network Abstraction Sub-layer for a SLIP connection.

The NAS_SLIP unit consists out of a slip unit and a UART as depicted in figure 5.2. The hierarchical description of the units within the SLIP_NAS unit allows reuse of these units. The SLIP unit contains the models of the slip encoder and slip decoder. The UART unit contains the logic to receive and transmit data over a basic three wire serial link. The serial clock is generated within the UART. Altering a constant of the VHDL model will alter the serial clock frequency.
The content of the IPv6 unit is depicted in figure 5.3. On the left-hand side the receiving units are depicted and on the right-hand side the transmitting units are depicted. The transmitting side gets its data from the TCP unit. The X_Bus_In unit contains the data source of the X-bus. The X_Bus_Controller generates the Xslot signal. The Xhandler2 unit is a model of an extension handler. The model of the extension handler does not contain a real extension header encoder, it just inverts the data signal. The IPv6 Encoder is used as output for the X-bus. The IPv6 encoder is a model of the described design. The outputs of the left-hand side can be connected to the inputs of the right-hand side. This reduces simulation-processing time.

The left-hand side gets a message, either from the NAS or the right-hand side. The IPv6 decoder processes the received message. The IPv6 decoder output (defined in figure 4.5) is put on the X-bus. The extension handler is also a dummy extension handler that just inverts the payload. When an error is found in the payload by the IPv6 decoder the invalid_message signal made high in order to signal a pipeline flush. When the IPv6 decoder discovers a parameter problem, it signals the ICMP unit. The ICMP unit then generates the ICMP parameter problem message.

The IPv6 unit does not contain a last minute length insertion unit or a configuration data unit because these were not modelled.
The content of the TCP unit is depicted in figure 5.4. The TCP unit contains 2 units called Basis_A, which absorb messages. The output D of Basis_A contains the last received byte. The other unit generates one message from one file. The file name is a constant that can be adjusted to send different messages.

The modelled units were all simulated before and after synthesis, routing and placement. Since the design is not completed, a prototype could not be developed. When the design is completed a fully functional prototype can be tested.

The test environment would consist of a microcontroller connected to the protocol stack and a PC connected to the stack with a serial link. First the transmitter should be tested, by programming the microcontroller to send test messages and analysing the results at the PC. When the transmitter is fully functional and conform the protocol specifications the transmitter can be used for testing the receiver. The microcontroller is then programmed to transmit the stack to the PC for analyses.

At the moment of writing the IETF is starting to develop test schemes for IPv6. These developments are promising and could be used for testing the IPv6 layer.
6 Conclusions and future work.

A web controlled device has to contain:

- A communication circuit to connect the device to the Internet.
- The IP protocol
- The TCP protocol
- The telnet protocol
- A textual command language to control the device

The protocol stack of the device has been implemented as dedicated hardware to reduce the load on the embedded processor. The organisation of a protocol stack encourages implementing the protocol stack as a pipelined data processor. The current design includes:

- A general interface to communication devices. The design of the web controllable device is therefore media independent.
- An IPv6 encoder and decoder to process IPv6 messages.
- A dedicated bus to enable the processing of extension headers in order of appearance in the message.
- A last minute length insertion unit to overcome the causality problems in generating not fixed length messages by a pipeline.

Although IPv6 is not an Internet standard yet, it has been chosen because it will become an Internet standard in the near future and its specification is not likely to change. IPv6 will support traffic classes and authentication, these developments will be very useful in web controlled devices.

Web controlled devices must contain a safety envelope to prevent dangerous situations as a result of the Internet delay. The safety envelope ensures the safety of the device by restricting the control to safe transitions between stable situations.

The design of web controlled devices is not complete. A lot of work has been done, but there is even more work to be done:

- When the IETF publishes the RFC on traffic classes, the use of virtual circuits defined in this RFC should be investigated and possibly a new class must be proposed.
- The modelling of Internet delay for controlled systems has to be investigated to develop web controlled devices in a regulating configuration.
- The communication of ICMP events between the ICMPv6 unit and all other units must be investigated.
- The structure of the unit that contains the configuration data should be investigated.
- The ICMPv6, extension handlers, TCP, telnet, configuration data unit and the interface to the embedded system must be designed.
- The IPv6 testing schemes should be studied when they are published by in a RFC.
7 References


[Rail Road] http://rr-vs.informatik.uni-ulm.de/rr/gui2/