Embedded Systems Conference East 2001
Class numbers 421 and 431

Embedded Internetworking with 8- and 16-bit Microcontrollers

Information provided is based on the material from a
2-day hands-on training class on Embedded Internetworking

By Olaf Pfeiffer
olaf.pfeiffer@esacademy.com

Embedded Systems Academy
www.esacademy.com

Embedded Systems Academy does not have a booth at the conference. However, you will most likely find one of us at our demo display in the Philips Semiconductor booth

What is Embedded Internetworking?

With "Embedded Internetworking" we categorize all embedded applications that require access to the Internet.

Some of the goals of Embedded Internetworking are to allow

- Anytime - anywhere - control of your electronic devices at home or wherever they might be
- Self-maintenance of electronic devices
- Pervasive computing
- Wearable computing

Why Embedded Internetworking?

Do we really need a toaster or coffee machine with web access? Which real benefits would we get from putting our appliances on-line?

Well, the answer is that everything needs to be put into the "right" perspective. Sure it's hard to believe that your regular toaster or coffee machine would gain substantial benefits from an Internet connection. However, there are circumstances where networked appliances have "real benefits".

The deregulation of utility services is one point. For more and more families, deregulation means that prices for power vary during a day and that power is more affordable during off-peak hours. Energy-hungry appliances such as washing machines, dryers and dishwashers could actively help to lower their operating costs, if they could recognize the off-peak hours themselves. If you do not need your clothes and dishes washed and/or dried immediately, you would just turn the machines on into a special energy costs saving mode called something like "Start your operating cycle as soon as you recognize an off-peak, lower-cost operating time".

Obviously, if features like this cost consumers more than they can potentially save, it will be a tough sale, unless there are additional benefits, for example self-maintenance. For anybody involved in software issues, the following scenario is not too far-fetched: The manufacturer of a washing machine recognizes, that a product that has been on the market for a year can be improved with a software update. The software executed by the microcontroller in the washer would need to be replaced. Sending a service technician to every home with such a machine would be too expensive. However, if that device has Internet access it could regularly check by itself, if its manufacturer has some updates and download them automatically.

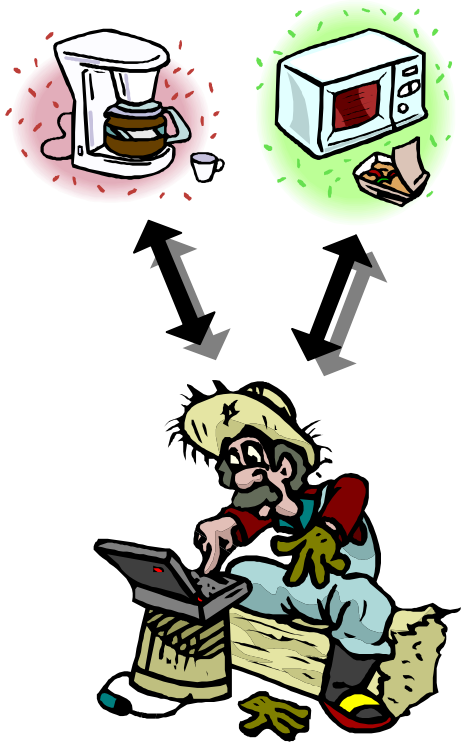
The benefits of any kind of network access for appliances become even bigger in any professional environment. Just as an example, let's look at "industrial" coffee machines for restaurants: the following networked features are already available today: Upon entering the order of a coffee into the register, a message is sent to the coffee machine to brew one cup of fresh coffee of a certain type and a specified strength. The machine can also send messages when it requires service, such as re-filling beans or cups.

Embedded Internetworking Applications

There are many potential applications for Embedded Internetworking. In general, most of these do not actually require the network used to be the Internet – it just eases using the application. When using the Internet, the end-user can access the embedded system with interface tools that are very familiar to him/her: like a web browser or email program.

Some potential applications

- **At home**
  - Meter reading
  - Sprinkler Systems
  - Security / Alarm Systems
  - Appliances
- **Scientific**
  - Remote data collection
  - Weather stations
  - Seismic recorders
  - Maritime: Buoys
- **Medical**
  - Patient monitoring devices
  - Homecare for the elderly and disabled
- **Automotive**
  - Fleet tracking



One application example would be fleet tracking for freight companies. There are already systems and networks in place that allow companies to track the individual cars and trucks of their fleet. Using the Internet instead of a customized network solution has the advantage that the data can easily be generated, transmitted, processed and viewed with standard tools. The truck can regularly send an email via a wireless modem. A server that puts the data automatically into a database receives the email. This database can be accessed with a regular web browser from any authorized person anywhere.

Embedded Internetworking might also be introduced to a user's home without him/her specifically requesting it or paying extra for it. For utility providers, the meter reading is a substantial expense. The original approach was to send somebody on a regular basis to every home to do the reading and enter it into a table or device. Some companies started to introduce wireless reading, where the reader just needs to "drive by" the home to get the reading. Today, the trend is to get the meter on-line so that it can automatically send its data via an encoded email to the provider on a regular basis.

Growth Potential

Eventually, the number of embedded electronic devices hooked up to the Internet will grow at a faster rate than the number of PC-style computers hooked up. This prediction is based on two facts:

1. Already today, embedded microcontrollers outnumber PC-style processors by far. Each computer comes with a variety of embedded microcontrollers in the drives, printers, modems, displays, keyboards and all other peripherals.
2. Even lower-end embedded microcontrollers get more and more powerful, allowing for enough processing power to handle networking tasks. In most applications, newer designs use a more powerful (faster or wider bus) microcontroller than the one used in the previous design. Networking functionality might already be built into the hardware (a free serial port could be sufficient) or can be added for a minimal additional cost.

So it is really just a matter of time when we will get to the point where the number of embedded controllers "on-line" outnumbers the number of PCs being "on-line".

Expected Services

Although the Internet provides a variety of different services, two play a dominant role: Web surfing and email exchange. For embedded applications, these two services already offer all the functionality required when it comes to accessing data and functions of embedded devices.

Email is a good service, if data needs to be transmitted on a regular basis and the transmission is not time critical. If the embedded device is primarily a data collector, like a weather station, that device could send an email on a regular basis (for example once per hour) with the latest sets of data.

If instant remote access to data and/or certain functions is required, web servers and browsers offer a better service. If the embedded device runs a web server, data and functions can be accessed instantly. The web browser could not only display data on dynamically updated web pages but also bring instant feedback to the device via forms.

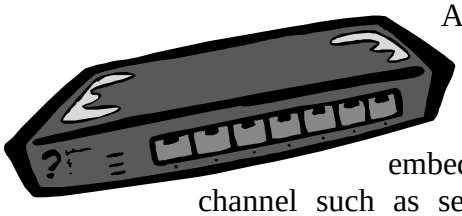
Challenges

The Internet supports many different protocols and connection methods. One of the first decisions that needs to be made for Embedded Internetworking devices is "Which protocols and applications do need to be implemented for this application?".

As Embedded Internetworking devices often do not have the CPU and memory resources to just handle the entire TCP/IP stack with all its variations, a careful evaluation of the different protocols and applications is required, to make a decision what is needed and what can be skipped.

The smaller and the more cost-effective an embedded internetworking device needs to be, the tougher it is to strip down the system to the bare essentials.

Can Gateways handle the communication load?



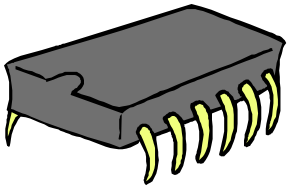
A popular way to minimize the burden on the embedded device is to not establish a direct Internet connection but to communicate via an external controller or gateway instead. The idea is to have the embedded device use a regular low cost communication channel such as serial interface, I2C or CAN to communicate to a gateway. The gateway itself has all the intelligence, CPU performance and memory required to implement an Internet connection.

There are many different implementations and variations of these kinds of gateways available today:

The biggest, most flexible gateways are implemented in pure software and run on regular PCs. These can get as small as embedded PCs can.

Even smaller, but less flexible gateways are available in "dongle-type". Designed into a small housing, most likely using a 16-bit microcontroller with sufficient memory, these devices can handle the entire TCP/IP stack and most available Internet protocols and applications. Such a device connects to the Internet via modem/phone line or might have an Ethernet connector. On the other side of the dongle, there are one or more serial connectors to communicate to embedded devices.

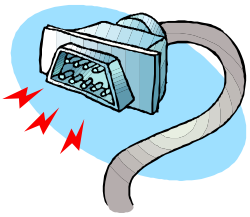
As you might have guessed, technology does not stop here – you can get gateways that are even smaller. There are "gateways on a chip", where the gateway functionality is embedded on a small PCB that fits into a PLCC socket.



Are there any standards?

One of the biggest challenges prohibiting fast growth of Embedded Internetworking on a large scale today is the lack of accepted industry standards.

1.) For homes with Internet access, we can almost make no assumption about the Internet service. As an embedded device seeking connection, do we use the phone line to dial-in? Which number? Which provider? Which communication parameters?



2.) There is no standard for a low-cost hardware interface. Ethernet is still too expensive to be adapted into low-cost, mass-market devices such as many kitchen appliances where the entire control circuit may just cost a few dollars.

3.) Once a standard would be available for the lower level communication, we would still have no standard in place about data objects and formats. If you replace a device with one from a different manufacturer, you will most likely get a completely different Internet interface (emails sent/received or web pages/forms used have a completely different format).

At the time of its publishing on April 1st 1998, the RFC# 2324 was just fun, a standard for a "Hyper Text Coffee Pot Control Protocol". Just a few years later it's suddenly much closer to reality than the author probably ever thought possible.

What about reliability?

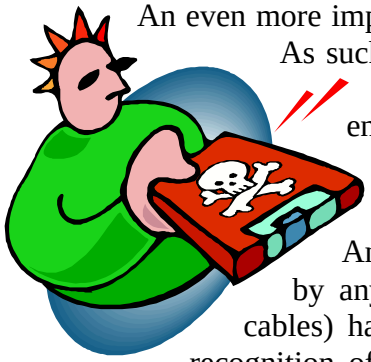
In general, embedded control devices do need to be much more reliable than our average PC. A software crash in the middle of the execution cycle of a washing machine or dishwasher is simply unacceptable. If an embedded device uses any functionality or feature involving the Internet, it must be designed in a way, that

a) a sudden, complete loss of communication is not fatal to the system and

b) it still operates within it parameters, even if messages have been intercepted and/or altered by intruders

Looking at the Internet service provided today, this might be the "real" challenge.

What about security?



An even more important aspect for embedded Internet applications is security. As such, the Internet is unsecured. When dealing with TCP/IP and the Internet, keep in mind that per default no encoding or encryption is done to the data packets transmitted. Most information is in plain ASCII, including transmitted login names and passwords.

Anything transmitted and received could have been intercepted by anyone. Every router (or any "sniffer" sitting on one of the cables) has access to all contents and could easily do an automatic recognition of certain keywords like "login", "password", "credit card", "social security number", "drugs", etc.

It's almost like a postal service without envelopes! Without additional encoding or encrypting mechanisms, the communication is open and accessible.

To secure transmission of private information, Netscape implemented SSL (Secure Sockets Layer) based on a 128-bit encryption code, which became a de-facto standard.

Unfortunately, handling a 128-bit encryption code is not really meant for an 8-bit or 16-bit microcontroller…

So what can we do with an 8051?
128-bit encryption/decryption?
Interfacing to a 16-bit Ethernet controller?

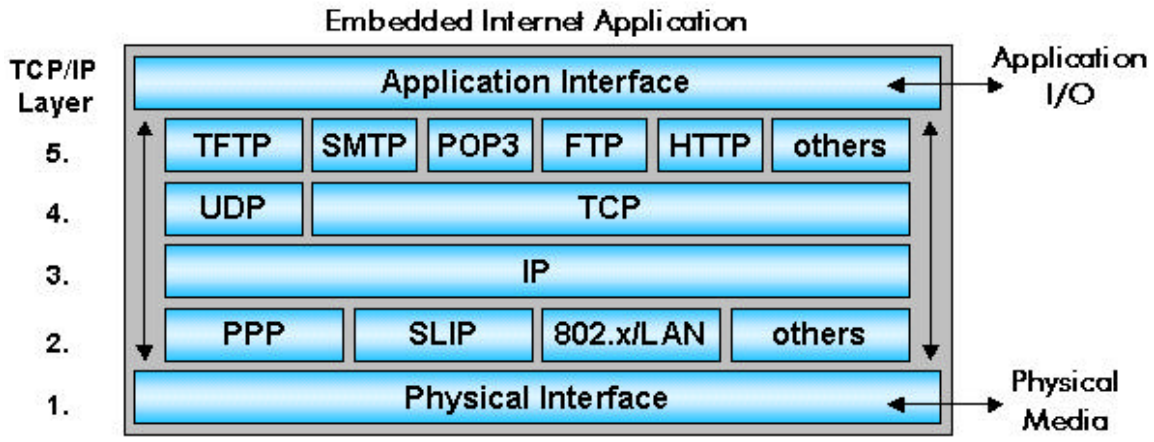
It sounds difficult, but gets easier with a specialized 8051 derivative.

Triscend offers an 8051 derivative with on-chip FPGA – programmable logic. One of the modules available for the FPGA allows the encrypting and decrypting of this code in hardware, relieving the 8051 core from that task.

Another module allows for direct interfacing to a 16-bit Ethernet chipset by providing a 16-bit wide data bus interface from the chipset to the 8051 core.

Summary of technical background information

The TCP/IP protocol stack



The TCP/IP layers are located in the middle of the entire communication stack. They ensure proper point-to-point communication and take care of the routing and delivery of packets.

Towards the physical media a device driver layer is inserted ensuring proper communication with the chosen physical interface.

Towards the application, we have an application protocol, which implements the requirements (usually a set of commands and replies) for a specific application.

To get any device hooked-up to the Internet successfully, at least one path through the entire communication stack is required. If the device supports multiple applications and/or multiple physical interfaces, several paths through the stack are possible.

Do we need all TCP/IP features?

When implementing the TCP/IP protocol stack to medium or low performance microcontrollers, we can simply skip some of the functionality usually used within the TCP/IP stack.

From all the functions provided, the fragmentation is one of the first candidates on the list that we can consider to drop completely. In the IP layer, messages that are longer than 64kbytes are fragmented into several pieces during transmission and re-assembled by the receiver.

On an 8-bit microcontroller with a total address range of 64kbytes, this is the kind of functionality you do not want to waste any of your valuable code space.

In any case, it needs to be ensured that the device generates an appropriate error response, if anybody tries to send a large, fragmented message to it.

IPv4 versus IPv6

The currently most often used Internet addressing is still IPv4 – using a 32-bit address, written as 4 decimal bytes separated by a dot. Such an address positively identifies one host (network node) in the Internet.

Example: `129.132.29.84`

As there are certain limitations on how these addresses are assigned to hosts, the Internet is running out of addresses and a new, extended system is required.

With the newer version - IPv6 - an address consists of 128 bits, written as 8 hexadecimal words separated by a colon. It was specified in a way that allows co-existence with IPv4, in order to keep the existing network operating and allowing for a slow transition to the new standard.

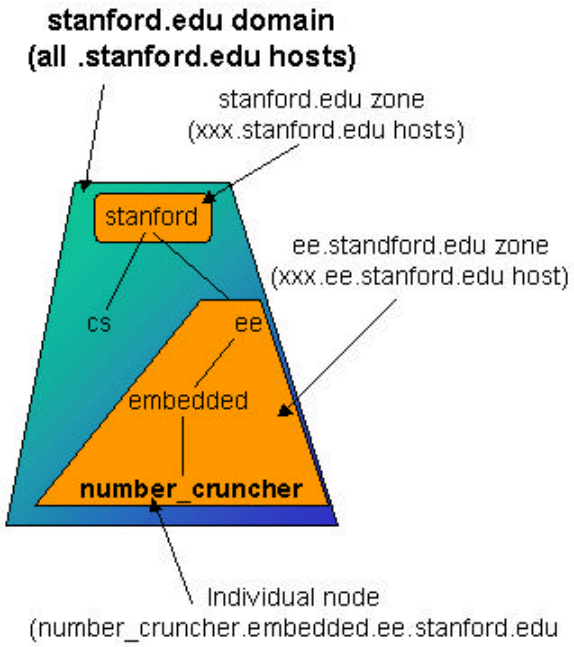
Example: `1234:5678:9ABC:DEF0:0123:4567:89AB:CDEF`

For many Embedded Internetworking applications, an implementation of IPv4 will be sufficient, as servers implementing DHCP can “share” one IP address with many nodes. DHCP stands for Dynamic Host Configuration Protocol, which dynamically assigns an IPv4 address to a connected host.

So if you only have one IP address assigned to your computer at home, but would like to connect several computers and other Internet-enabled devices, you can “share” this one IP address by installing a local server with DHCP (for example integrated in a gateway). The server assigns completely different IP addresses to the locally connected hosts and automatically translates all requests from local hosts on the LAN to the routers on the WAN, using only one address for all WAN requests.

Domain name support

Domain names are a way to increase readability of IP addresses. The domain name system allows mapping a readable name like www.esacademy.com to an IP address. So-called domain name servers operated by the Internet Service providers implement and maintain the database involved in this mapping process.



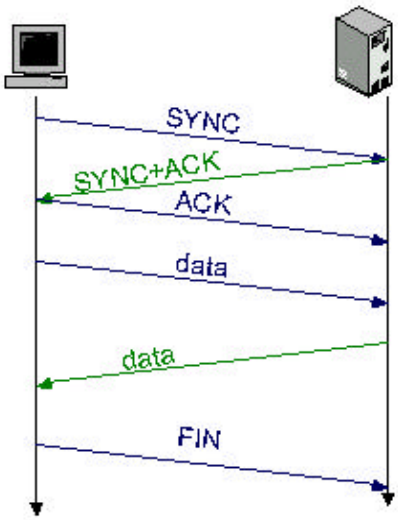
For Embedded Internetworking applications it needs to be evaluated how important a fixed domain name is:

If the application provides a service where somebody needs to be able to log in from an external location (like any server type application – web, FTP), a fixed address is required. Only a fixed address allows for a fixed domain name.

However, if a DHCP server assigns the address to the device dynamically, a fixed name can currently not be supported, meaning that the device cannot be a server (for example for web pages of file transfer).

The three-way handshake of TCP/IP connections

To get a feeling for the network traffic created, let's briefly look at the communication involved for a TCP/IP connection.



1.) Client sends a segment with SYNC bit set and ACK bit cleared in flags sequence number contains a random sequence start number to avoid confusion with other transmission

2.) Server replies with a SYNC and ACK bit set

3.) Client confirms connection with only ACK set and the connection is established

4.) Data is exchanged for as long as needed

5.) Any partner can terminate connection using the FIN flag

Keep in mind that any part of the communication above will most likely pass several routers (each one completely receiving a packet before forwarding it). If the data is too long it will be fragmented into pieces – each piece might travel different routes – and the receiver needs to re-assemble the data.

What is required to send/receive email?

Email is one of the basic services of the Internet. As passing along the net, each email is handled by mail-demons. These are servers on the Internet managing the transmission of email messages. Each demon will entirely receive an email and store it in a local buffer before deciding on what to do with it and if and where to forward it to.

That is why email delivery can take anywhere from a few minutes to sometimes hours or even days, as the demons can keep mails in their buffers for days after a delivery failure.

Encoding of email

Email is encoded using the standard ASCII characters only (7-bit), no encryption or compression.

Binary attachments in MIME format are translated to the valid ASCII characters as follows:

1.) Divide the binary contents of three bytes (from the binary attachment) into 4 groups of 6 bits.

2.) Each 6-bit group is coded into a valid ASCII character of the email.

3.) Repeat until end of data.

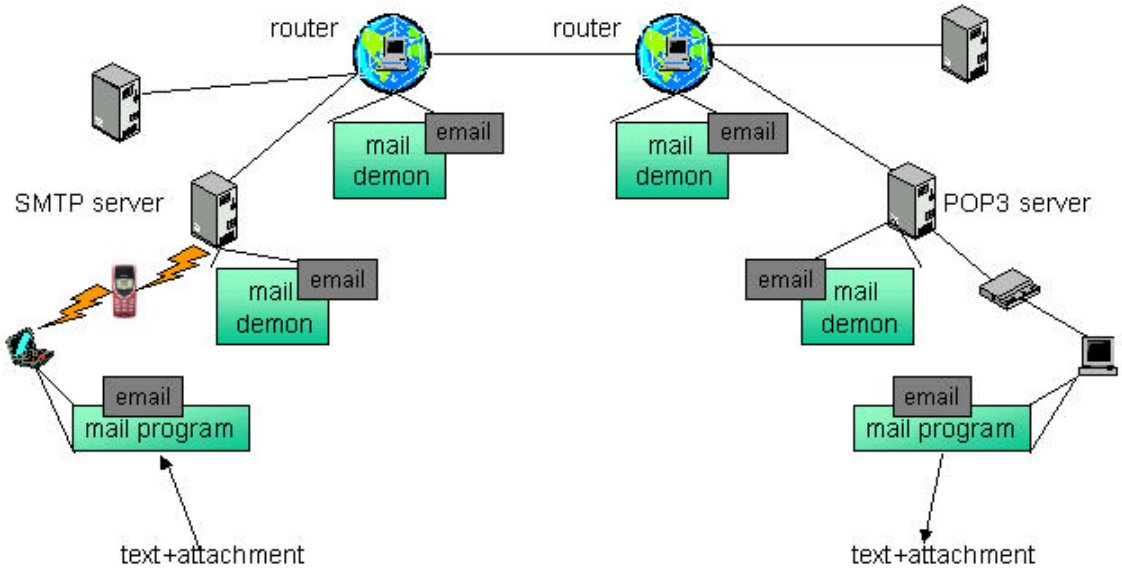
Although this way binary attachments are supported, it is a very inefficient way to transmit data, as the email will be about 33% longer than the binary file transmitted.

Accessing the servers

There are two main protocols involved in email delivery: SMTP and POP3.

- Using SMTP, a host transmits mail using simple commands like HELLO, MAIL FROM, RCPT TO, DATA and QUIT
- Using POP3, a host receives mail using simple commands like LIST, RETR (retrieve), DELETE and QUIT

Illustration



In the illustration shown, an email with text and attachment is encoded and sent to an SMTP server for further transmission. The SMTP server evaluates the email address of the recipient and forwards the email to the demon. En route, the email is handled by several demons until it arrives at the POP3 server of the recipients email account. Upon next login, the recipient's mail program checks the POP3 server and downloads the email.

During the process, each demon adds an entry to the header of the email with information about when and how he handled the email for diagnostic purposes.

What is required to request/serve web pages?

HTTP – Hyper Text Transfer Protocol summary

- Based on a TCP connection between the client running a browser and the port assigned to HTTP on the server
- Uniform resource locators (URLs) identify pages:

  `http://host.domain/directory_tree/filename.ext`
- Client and host exchange messages with ASCII text commands like GET, HEAD and POST

HTML Basics

The default format for requested files is HTML (Hyper Text Markup Language). HTML is limited to the standard ASCII code (7-bit) and inserts tags to add formatting information – or to "markup" the text.

The characters "<" and ">" encapsulate the tags.

Most tags are marking the beginning and the end of a text section. The end is marked with the tags "</" and ">".

Examples:

```
<p>Marks a paragraph</p>
<b>Text to be displayed bold</b>
<i>Text to be displayed in italics</i>
```

A minimum HTML file requires the tags HTML, HEAD and BODY:

```
<html>
  <head>
  </head>
  <body>
    <p>My paragraph</p>
  </body>
</html>
```

- The header part may contain information about the document, like title, keywords, author, date, expiration date and others. A browser can request to only get the header of a document using the HEAD command.
- The body part contains the actual text to be displayed in the browser.

For embedded applications, it is important to be able to dynamically modify the contents of a web page. (If it would be static, why would we want to put on an embedded device in the first place.)

The dynamic insertion of text into a requested page is supported via SSI – Server Side Inlcudes. SSI use special tags to inform the server that they need to be replaced with something else.

The tag for SSI calls looks as follows:

```
<!--#exec cgi="FunctionName"-->
```

If a web browser requests a page containing such a tag, the following happens:

1.) The web server starts copying the contents of the web page to the output buffer.

2.) While copying, the server watches for SSI tags.

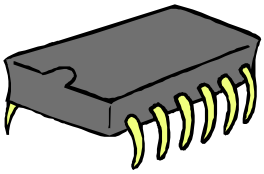
3.) If a SSI tag is detected, the function named is located and called, passing a pointer to the current output buffer.

4.) The function can now dynamically insert text into the output buffer, by just copying ASCII characters directly into the buffer.

5.) When completed, the functions returns, passing the number of characters inserted back to the server.

6.) The server continues to copy the web page to the output buffer until either another SSI is detected (and the process 3, 4, 5 and 6 repeated) or the end of the page is reached.

Running a web server on an embedded microcontroller



If the web server runs on an 8-bit or 16-bit microcontroller, the following should be considered for a minimal implementation:

- A virtual file system is required
  - As there is no hard drive, web pages and other static portions of the files need to be stored in ROM
  - Offers an easy way to integrate these files into the application
  - Handling of directories and sub-directories is not necessarily required, if all incoming requests can be assumed to go to a "root" directory
- Memory consumption
  - The transmit buffer in RAM must be at least as big as the biggest web page ever served
  - The ROM must be big enough to hold all static files served
- Server side include supported
  - This functionality allows to dynamically modify contents of web pages to display values of inputs
- Forms supported
  - This functionality allows to upload data via forms to the web server, so that changes to outputs or uploads of data files are supported

The following functionality will most likely NOT fit into such a minimized version of a web server:

- Full-featured file system
  - Providing several directories and sub-directories
- Cache for served files
  - Used to speed up access to frequently served files
    - Not required in Embedded Web Server, as all files are already in RAM/ROM for fastest access
    - Pages with dynamic contents should not be cached anyway.

- Log files for access statistics
  - Logging statistics requires additional storage space that might not be available in an embedded application
  - A typical, single log entry can add up to more than 200 bytes
    - Date & Time
    - URL of file served
    - User's PC identification
    - Referring URL

Minimize the ROM requirements by co-locating static files:

- If the embedded web server is connected to the Internet full-time, co-locating certain static files to a regular web server can save resources on the embedded web server
  - Keep all static pages, graphics and other files on a regular web server. (Exception: the main entry page MUST be on the embedded server)
- There are two ways to implement dynamic pages:
  1.) Implement fully on embedded web server, but have all graphic elements come from a regular web server.
  2.) Have the dynamic page hosted on a regular server, only calling a cgi-function (in general, this is like directly calling the SSI function discussed earlier) on the embedded web server. This way the embedded web server does ONLY serve dynamic data.

Embedded Internetworking with firewalls

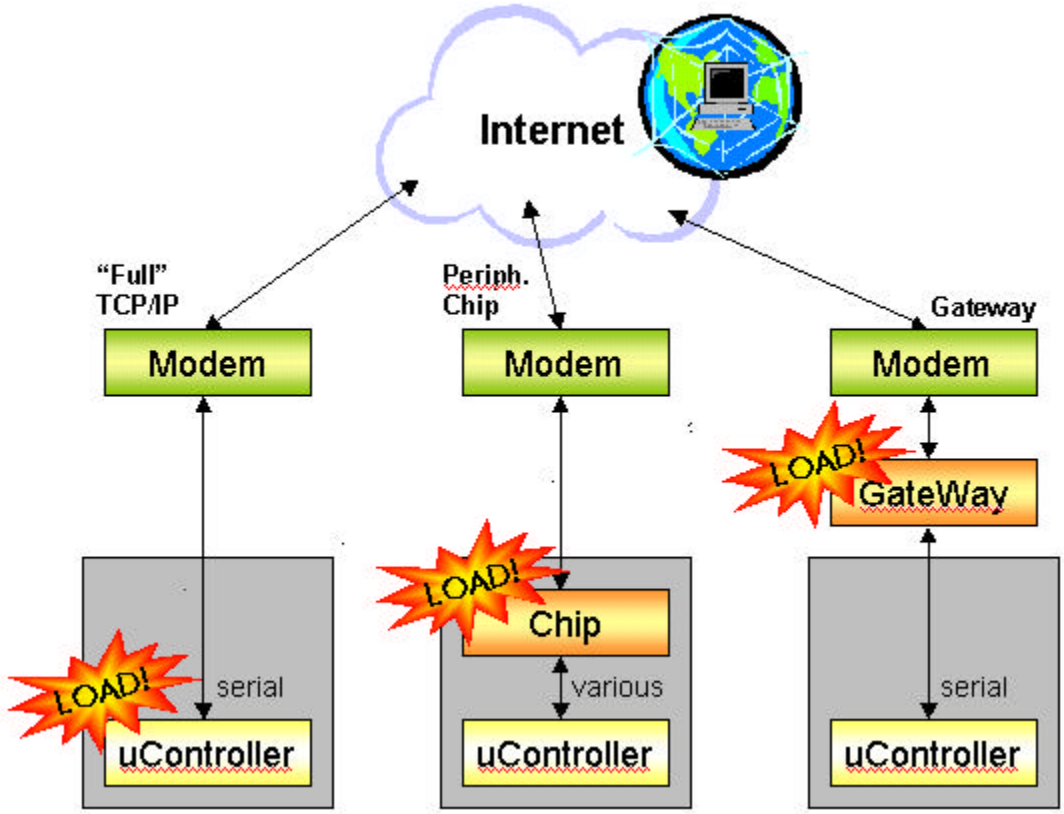
The idea of an Internet Firewall is to have protection from un-wanted or un-secure requests coming from the Internet (WAN) to the local network (LAN). Firewalls are implemented by physically disconnecting the LAN from the WAN - and putting the Firewall in between. To do that, a firewall requires at least two network connections, one for the WAN and one for the LAN.

Every single TCP/IP package coming from the WAN will be examined and only forwarded to the LAN if it is considered "safe".

In default configuration, many firewalls do not allow requests to local servers (no matter if web/http, ftp, or other) but allow users from inside the LAN to access all standard, external services.

A customized, direct point-to-point connection from the LAN to an embedded device would not be supported by the default configuration of firewalls. If an embedded device implements some server functionality (like http or ftp) and is behind a firewall, the firewall must be configured to allow access to the device, otherwise it will be inaccessible from outside the LAN.

Available solutions for Embedded Internetworking



There are a variety of different commercial solutions available today that allow you to connect your embedded device to the Internet.

In general, we can differentiate the following categories:

1.) TCP/IP implementation on the microcontroller of the application

2.) TCP/IP implementation on a peripheral chip (gateway-on-a-chip)

3.) TCP/IP implementation in external gateway for single devices

4.) TCP/IP implementation in external high-profile gateway

The main difference between 3 and 4 is, that gateways in the category 3 are meant for one embedded device. The gateways in category 4 can handle many embedded devices and might have additional functionality, like taking care of security issues, by also implementing a firewall.

Let's compare these different implementation concepts:

1.) TCP/IP implementation on the microcontroller

Solutions in this category implement the entire TCP/IP stack in software – running on the application's microcontroller. In order to be able to do this, an appropriate hardware interface is necessary, such as a serial port communicating to a modem or an Ethernet chipset for direct Ethernet access.

PROS:

- Enables an embedded device to "directly" hook-up to the Internet
  - Requires no additional hardware other than the modem, no gateway or black box needed
- Hardware modifications to embedded device are minimal
  - Assuming following resources are available
    - Memory requirements
    - Physical hardware interface (e.g. serial port to modem)
- Plain software solution
- Available for virtually all microcontrollers
- No customized, proprietary protocol required (like with some gateways)

CONS:

- Uses the main controller of the embedded application to handle the Internet communication
  - How much code and data space is needed?
  - How much CPU time is needed?
  - Security issues? Can the CPU deal with all security issues or is an additional firewall required?

2.) TCP/IP implemented on a peripheral chip

From this category on, we always have some sort of gateway between the application's microcontroller and the Internet. The main differences are where this gateway is located (within the application or outside) and how it is implemented.

Solutions in this category either implement the TCP/IP stack handling "in hardware" on customized chips or simply try to build the smallest possible gateway on miniature single board computers that can be integrated into an application.

One of the smallest ones is implemented on a PCB the size of a PLCC68 package, so that it can actually be inserted into a PLCC68 socket.

PROS:

- Enables an embedded device to "directly" hook-up to the Internet
  - Requires no additional hardware other than the modem, no gateway or black box needed
- All Internet related connectivity is implemented on one peripheral "chip"
- Minimal requirements on the CPU load of the applications' microcontroller
- Minimal requirements on the application's memory
  - The peripheral "chip" has its own memory buffers

CONS:

- Hardware modifications required
  - The Internet chip needs to be designed into the application
  - How difficult is it to make these modifications?
- As with every "chip" vendor, evaluate carefully the long-term pricing and availability of the Internet chip
  - Second source available?

3.) TCP/IP implemented in an external gateway for a single device

The gateways in this category are implemented in their own housings and are meant as an external gateway for single embedded devices. The housing is usually quite small (roughly the size of a mobile phone) and has just a few connectors: towards the embedded device a regular serial port and towards the Internet either a jack to the regular phone line or an Ethernet connector.

PROS:

- Requires only a small amount of the resources from the embedded application
  - A C-library needs to be incorporated to the application to allow access to the gateway's functionality
- No hardware modification of embedded application required
  - Assuming the serial port is available
- Minimal requirements on the CPU load
- Allows for a short time-to-market, as no hardware modifications are required and software modifications are minimal

CONS:

- Requires one gateway for each embedded device
  - Costs?
- In general, gateways are too expensive to be only used by single devices, especially in low-cost applications
- No standardization of protocol between embedded device and gateway, each manufacturer has a proprietary protocol
- How does the gateway handle the security issues?

4.) TCP/IP implemented in an external high-profile gateway

The gateways of this category are powerful enough to handle multiple embedded devices and to take care of security issues. They are either implemented in software running on a PC or are integrated into a stand-alone gateway, usually based on an embedded PC.

PROS:

- Requires only a small amount of the resources from the embedded application
  - A C-library needs to be incorporated to the application to allow access to the gateway's functionality
- No hardware modification of embedded application required
  - Assuming the serial port is available
- Minimal requirements on the CPU load
- Allows for a short time-to-market, as no hardware modifications are required and software modifications are minimal
- Multiple embedded devices can share one gateway

CONS:

- Requires one gateway for each embedded device
  - Costs?
- In general, gateways are too expensive to be only used by a few single devices, especially in low-cost applications. For each application, consider: How many gateways are needed and can be shared by how many embedded devices?
- No standardization of protocol between embedded device and gateway, each manufacturer has a proprietary protocol

About residential gateways

A category not yet mentioned comes from the "other end" of the Internet connection:

All solutions we have shown so far come from companies in the embedded marketplace, trying to expand their embedded devices into the Internet.

Companies primarily involved with providing the Internet infrastructure want to get "closer" to the embedded applications, too. The solutions driven from this end focus more on the problem on how to provide multiple standardized Internet access ports to each home via "Residential Gateways".

Although there were several standardization committees founded, there is no real industry standard in sight yet. Too many companies have to many different interests on where and how to implement these residential gateways.

Some would like to see it integrated with the cable or DSL modem, some want it to be pure software running on a dedicated PC and others prefer it to be in the audio/video rack as the multi-media access port to your stereo and TV.

So which solution is best for what application?

Unfortunately, we cannot give a generic answer to this question – it always depends on all inputs and circumstances of a specific application. All we can give here are a few rough guidelines.

If your application is low-volume and consists of stand-alone embedded devices (not clusters of embedded devices), solution 3 “single gateway for single device” might be a good choice, especially if you need to get your product to market fast.

If the volume gets higher and/or the systems cost is a bigger issue, solution 1 “implement TCP/IP on application’s MCU” or 2 “TCP/IP on a peripheral chip” will most likely be a better choice.

As soon as multiple embedded devices can be clustered (for example all appliances in the kitchen or all the equipment in your audio/video cabinet), solution 4 “high-profile gateway” is the first one you should consider.

Examples of commercial solutions

Micronet by CMX

Micronet is a minimized TCP/IP protocol stack offered by CMX, optimized for 8-bit and 16-bit microcontrollers. Options to implement a web server or mail services are available.

On an 8051, Micronet has the following memory requirements:

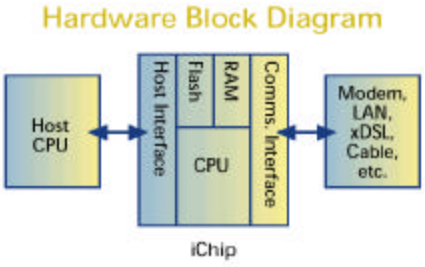
	ROM (app. In bytes)	**RAM (app. in bytes)**
Core + TCP/IP	4,650	100
PPP	3,750	100
Modem	450	50
HTTP (web server)	1,750	100
Virtual File System	600	50
TOTAL	11,200	400

Note that the calculation above does NOT include a single web page and does NOT include the buffers for transmit and receive. Both buffers need to be as big as the biggest TCP/IP data message that the application needs to be able to transmit or receive.

NOTE: A live demo of a Micronet web server implemented on a Philips 8051 microcontroller starter kit by PHYTEC is shown during the class and during the conference at the Philips booth.

iChip by Connect One

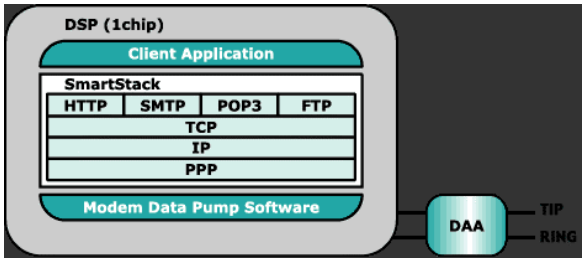
The iChip is an “Internet chip” that acts as a peripheral chip integrated into your hardware. Although implemented on a PCB, it is made of the right size to fit into a PLCC68 socket. The chip consists of a 16-bit microcontroller with Flash memory and RAM. The entire TCP/IP stack is implemented in the chip’s firmware.



The chip offers a selection of connection interfaces to both the application's microcontroller (serial or parallel) and the Internet (modem or Ethernet).

In general, the iChip also supports Flash programming. A demo implementation uses the Philips 16-bit XA G-49 with on-chip Flash. In that demo the iChip can re-program the Flash memory of the microcontroller. It allows receiving new code for the target hardware as email attachments. Simply attach a hex file to an email with a special subject line (password) and send it to the board.

SmartStack by e-Device

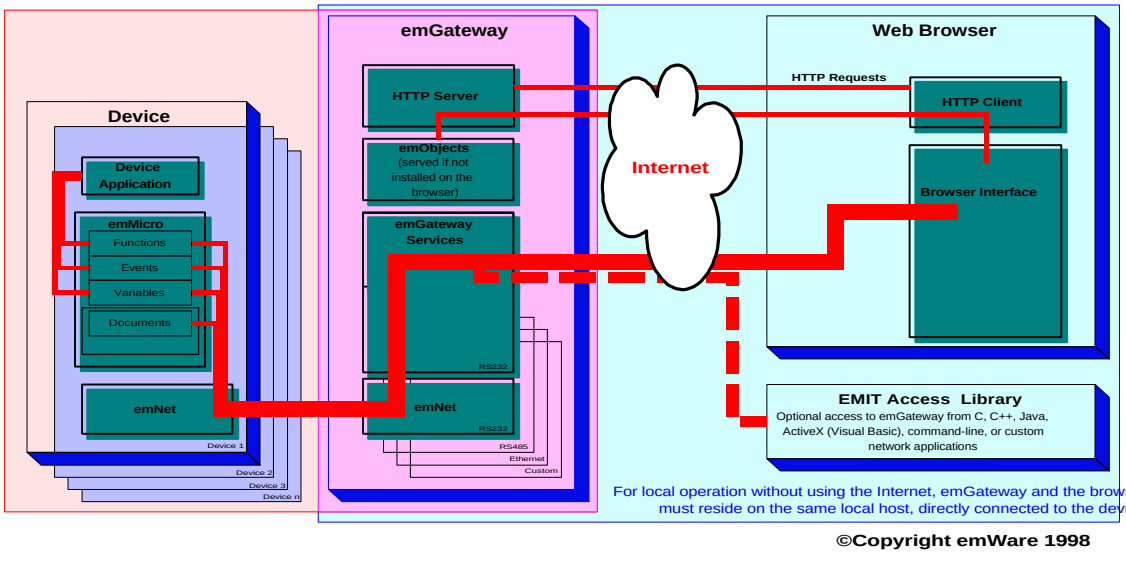


SmartStack is a DSP implementation of TCP/IP. The big advantage of using a DSP is that a DSP can also implement a modem. So all major components of the system are provided by the DSP: modem, PPP and TCP/IP stack including application service like HTTP for a web server or SMTP and POP3 for email handling.

Depending on the application, the DSP could also handle the tasks of the application, making this almost a single-chip implementation.

EMIT by emWare

The company emWare offers a variety of Internet enabling solutions based around EMIT (Embedded Microcontroller Internetworking Technology). EMIT is a C-library available for many different microcontrollers that implements the communication protocol required by the gateways of emWare. Using the library, embedded devices can use a standard serial channel to communicate to a gateway that handles all of the TCP/IP communication.



Only a minimum of additional functionality needs to be implemented into the embedded device, as most of the workload is handled by the gateway.

Summary and Outlook

As outlined before, Embedded Internetworking has a tremendous growth potential. The technologies involved and required to "make it work" are not exactly "rocket science" and many commercial solutions to Embedded Internetworking are available today.

One of the biggest challenges is the lack of a dominant standard for Embedded Internetworking. Anybody trying to implement an Embedded Internetworking faces basic questions like: What kind of Internet connection will most of my customers have? Do I need to dial-in via modem (where to, which local number, which communication parameters) or is there an "always-on" Ethernet connection available nearby?

As there are so many different companies from many different business areas (appliances, consumer electronics, internet service providers, etc.) involved in the development of Embedded Internetworking solutions, it is unlikely that we will see a dominant standard for Embedded Internetworking solutions any time soon.

It is more likely that we will see several application specific solutions evolve in parallel:

- We currently see a constant growth of always-on, high bandwidth Internet service solutions (cable-modem, DSL and others). These provide the user with at least one IP address.
- The next step is, that these users realize that in the long run they need
  a) Better protection from intruders using a firewall
  b) More IP ports to share the Internet connection with several computers and Internet enabled devices

  Combination products of firewall and DHCP server to share one IP address with multiple devices are already on the market for less than $100 (for example Barricade by SMC).
- Once the customer base with solutions like the one above reached a certain threshold, manufacturers thinking about devices with Embedded Internetworking can assume that a fair amount of end-customers will have a DHCP server in-house that can assign an IP address dynamically. A standardized Internet access port – so to speak.
- At that point it makes sense for companies to develop application specific solutions relying on the availability of that access port. We could see examples like the following popping up:

  - Companies in consumer electronics might build a common gateway for your audio/video cabinet with an Ethernet connection to your DHCP server on one side and something like USB or Firewire towards the audio/video equipment.
  - Companies in kitchen appliances might build a common gateway for all kitchen appliances. The gateway could be part of a kitchen computer as a shared input/output device for all appliances.

More information: web links

The Embedded Systems Academy has a collection of all links related to Embedded Internetworking available at

www.embeddedinternetworking.com

Links of companies and or products mentioned in this paper/presentation:

CMX
Micronet TCP/IP stack

www.cmx.com

e-Device
TCP/IP stack for DSP

www.edevice.com

EmbeddedLinks
Database of links relevant to the Embedded Marketplace and event calendar

www.embeddedlinks.com

Embedded Systems Academy
On-line training classes, multi-day training classes on Embedded Internetworking, consulting services for Embedded Internetworking

www.esacademy.com

emWare
EMIT protocol and emGateway solutions

www.emware.com

Philips Semiconductors
8-bit and 16-bit microcontrollers

www.philipsmcu.com

Phytec
Single board computers that can operate the TCP/IP stack

www.phytec.com

SMC
Broadband router, firewall and DHCP server

www.smc.com

Triscend
Microcontrollers with on-chip FPGA, allowing for 128-bit encryption in hardware.

www.triscend.com