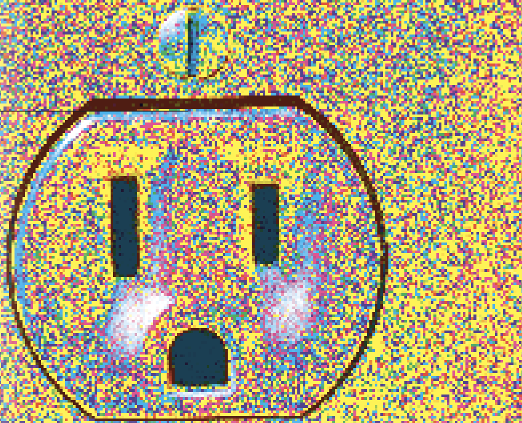
By Nicholas Cravotta, Technical Editor



INTERNET-CONNECTED DEVICES ARE NOT ALWAYS MORE INTELLIGENT THAN STAND-ALONE DEVICES. CONNECTIVITY MEANS MORE INTELLIGENT DEVICE MANAGEMENT.

Embedded TCP/IP: a smorgasbord of options

TCP/IP, A WIDELY DEPLOYED STANDARD, lets you connect and control devices and communicate with software on almost every operating system over most transport media. A key to embedded connectivity is the infrastructure. Most workplaces have Ethernet networks to plug devices into. In homes, power- and phone-line media are available. You no longer need to provide an end-to-end path between devices and a control server; instead, you can just hook the device into the infrastructure. In environments lacking an infrastructure, wireless technology can solve most problems. You can now access connectivity for embedded devices with little added cost.

Connecting to an infrastructure is also fairly straightforward. If the connection is to a LAN, the device gets the LAN's hard or soft IP address. Using a phone line, the device can use caller ID to determine whether an incoming call is for it. If so, the device can connect the call and allow itself to be queried, and, when the line is not in use, the device can periodically check in.

Pie-in-the-sky examples of Internet usage exist: Yes, a company is developing an "Internet teapot," so that adults with older parents will be able to see whether Mom and Dad are OK because they're still brewing tea several times a day. Internet-enabled light bulbs seem silly at first glance, but you probably wouldn't put Internet connectivity in the replaceable light bulb itself but rather in the permanent socket. Connectivity makes sense for high-end lighting, in which control and monitoring can justify the added cost per node.

Some compelling commercial applications focus on remote monitoring services, such as vending machines, security devices, and even espresso machines. Espresso machines, for example, can e-mail their leasing companies with status updates, such as fuse or temperature problems. The machine also tracks inventory and usage, automatically calling for a shipment of coffee as needed. Give a device a TCP/IP stack, and you can take care of that device from anywhere in the world.

FAT FREE

A common way of implementing TCP/IP is by selecting an RTOS that supports TCP/IP. QNX's Neutrino RTOS, for example, has TCP/IP modules enabling either a full or an "embedded" stack. The embedded version eliminates components, such as forwarding between interfaces, a complete Web server, and e-mail support, which embedded devices generally don't need. The smaller embedded stack also limits how you can configure it and what sockets it can use, although most TCP or UDP applications can run on either stack. QNX's stack is modular, meaning that you can dynamically load it when necessary for memory-constrained devices. Additionally, because Neutrino requires no binaries linked to the kernel, you can update your system in pieces, thus potentially allowing remote reprogramming or upgrades without rebooting.

When looking at an embedded stack, examine what makes an embedded stack different from a full stack. For one thing, an embedded stack is based on a proprietary socket's API. Many vendors port the Berkeley socket stack and call it embedded, but the Berkeley stack comes with many desktop features that don't necessarily apply to the embedded world. A proprietary stack provides more efficiency, but the trade-off is a slightly different API.

You also have the option of employing UDP, depending upon how reliable a communication channel you need. For example, TCP ensures that data arrives intact by employing a checksum and retransmitting any lost or corrupted data. TCP also guarantees in-order arrival of data at the application layer; the node receiving the data restores the order. UDP, in contrast, has an optional checksum, no in-order guarantee, and no retransmission, although an application itself can simulate these functions. UDP is useful when you have a reliable transmission media, such as a LAN, and you don't want the complexity or overhead of TCP. However, software implementations can require a lot more memory than do low-end processors. For example, a 40-kbyte stack may seem small until you try to squeeze it onto a 64-kbyte processor.

HOUSE SPECIAL

Given the maturity of TCP/IP, it's unsurprising to find vendors that have implemented the TCP/IP stack in hardware. Many devices, such as credit-card machines, vending machines, and meters, that connect to the Internet use processors lacking the ability to implement TCP/IP. However, using a chip or core such as one from iReady allows even 4-bit devices to connect directly to the Internet. These chip implementations can also consume less power than general-purpose CPU implementations. Seiko based the $7 (volume quantity) S7600 chip on the iReady core. To prove the concept, iReady added a Microchip CPU to a 1-kbyte Web server to make an Internet-enabled light. You can turn the light on and off at www.mycal.net/wsweb. The core has 40,000 to 225,000 gates, based on the transport method. The 225,000-gate version includes an Ethernet MAC. When a 56-kbyte modem drives the chip, it uses an average of less than 1 mW, making it attractive for both wireless and cellular applications.

You can also choose a DSP-based TCP/IP stack, such as the SmartStack from eDevice running on an Analog Devices ADI218x DSP. Using a DSP lets you put the modem software and the TCP/IP stack on one chip. Additionally, if all you need is connectivity, you can chose one of the less powerful and less expensive members of the ADI218x family, or, if you need a few DSP MIPS, you can select a member with a faster clock and more memory. You could use those extra MIPS, for example, to play back MP3 files or voice messages that you download to the device. Prices for the SmartStack/ADI218x chip set start at $18.50 (10,000), including the analog-front-end chip. Future versions of the chip set will include different connectivity options by integrating Ethernet or wireless MACs using an external physical-layer device. Thus, the same design code and DSP could serve different connectivity infrastructures by selecting a different interface.

Deciding whether to go with hardware or software, full stack or proxy, requires a thorough understanding of how your device will connect to the Internet, what kind of information it needs to pass and receive, how easily you can integrate the software or chip into your design, and whether adding the stack will require a complete redesign of your product. For applications with a powerful processor, supporting connectivity is a matter of

AT A GLANCE

▷ Connecting devices to the Internet enables remote control and monitoring of those devices.

▷ Embedding a TCP/IP stack in a device allows you to control and monitor that device from any computer in the world.

▷ TCP/IP stacks are available as software libraries, integrated RTOS components, and dedicated hardware ICs.

▷ TCP/IP is not appropriate for all applications. Through proxy protocols and gateways, low-cost devices can also connect to the Internet.

ACRONYMS

API:	application-programming interface
ARP:	Address Resolution Protocol
DHCP:	Dynamic Host Configuration Protocol
DNS:	domain-name server
FTP:	File Transfer Protocol
LAN:	local-area network
MAC:	media-access controller
POP3:	Post Office Protocol 3
RTOS:	real-time operating system
SMTP:	Simple Mail Transfer Protocol
SNMP:	Simple Network Management Protocol
SOHO:	small-office, home-office
TCP/IP:	Transmission Control Protocol/Internet Protocol
UDP:	User Datagram Protocol

adding a software stack and network interface. For applications riding a bit tight on memory or MIPS, using a proxy protocol with a gateway or implementing the TCP/IP stack as hardware is probably the way to go.

À LA CARTE

Some applications cannot handle the demands of a full TCP/IP stack. The main processor in many applications is perhaps an 8- or a 16-bit microcontroller with insufficient headroom for a complex stack. Internet-fitting such applications requires either adding a second processor, which adds cost; redesigning the original 8- or 16-bit design, adding cost and time; or using a proxy protocol to bridge to TCP/IP.

For example, you might want to use the Internet to control the lights in your house or your thermostat. However, the cost of a TCP/IP light bulb and the added expense of a thermostat might be prohibitive. Additionally, the messages a light bulb receives, such as on, off, and dim, require only a few bits of information, yielding TCP/IP-connection overkill. Instead of directly connecting the bulb to the Internet, you could instead use a proxy protocol to communicate between the bulb and a gateway and let the gateway translate messages to TCP/IP, thus bridging the gap to the Internet.

SOME COMPANIES MIGHT OBJECT TO THE REQUIREMENT FOR A GATEWAY ON THE NETWORK, BUT, EVEN IN APPLICATIONS WITH FULL TCP/IP SUPPORT, A GATEWAY STILL PROBABLY ADDS ENOUGH VALUE TO BE A REQUIREMENT.

Use of a proxy is appealing to companies, such as appliance manufacturers, that shave pennies and cannot justify upgrading a CPU just to add networking, especially considering that networking is a tenuous feature at best for white goods, such as refrigerators and dishwashers. However, such companies can squeeze a lightweight proxy into a system. For example, emWare's EMIT (Embedded Micro-Internetworking Technology) proxy stack is 1 kbytes in assembly and can cost less than $1 per device, including a bundled gateway, device software, and client-side tools.

Proxies, however, require a gateway device in the network to bridge the proxy protocol and TCP/IP. For example, a simple application, such as an Internet-enabled meter, does not require a full TCP/IP stack to manage the few bytes of data that it exchanges with a control server. The gateway takes care of the task of converting the proxy protocol into proper TCP/IP and back again.

Note that a gateway can take many forms—from a device on the local LAN itself, to a service that an ISP or portal supports, to an implementation on the destination server. At the appropriate stage, the gateway would recognize and route proxy traffic to a proxy server; the proxy server then converts the traffic to TCP/IP and reintroduces it into the data stream.

Some companies might object to the requirement for a gateway on the network, but, even in applications with full TCP/IP support, a gateway still probably adds enough value to be a requirement. For example, your data channel might use

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, go to our information-request page at www.rscahners.ims.ca/ednmag/. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

Accelerated Technologies
www.atinucleus.com
Enter No. 306

Analog Devices
www.analog.com
Enter No. 307

ConnectOne
www.connectone.com
Enter No. 308

Datalight
www.datalight.com
Enter No. 309

eDevice
www.edevice.com
Enter No. 310

emWare
www.emware.com
Enter No. 311

Ipsil
www.ipsil.com
Enter No. 312

iReady
www.iready.com
Enter No. 313

iReady Developer's Forum
www.iready.org
Enter No. 314

Lantronix
www.deviceserver.com
Enter No. 315

Microware
www.microware.com
Enter No. 316

Precise
www.precise.com
Enter No. 317

QNX
www.qnx.com
Enter No. 318

Rapid Logic
www.rapidlogic.com
Enter No. 319

Seiko
www.seiko.com
Enter No. 320

Ubicom
www.ubicom.com
Enter No. 321

WindRiver
www.windriver.com
Enter No. 322

Zilog
www.zilog.com
Enter No. 323

SUPER INFO NUMBER
For more information on the products available from all of the vendors listed in this box, enter No. 324 at www.rscahners.ims.ca/ednmag/.

an inexpensive, low-bit-rate modem or a wireless connection and thus be too narrow to support TCP/IP. Additionally, for some applications, such as home networks, several protocols may be in use. For example, nodes on a home network might use TCP/IP; power-line protocols, such as X10; phone-line protocols, such as HomePNA; and wireless protocols, such as 802.11, HomeRF, Bluetooth, or cellular. A gateway would serve as a bridge between all of the protocols. Finally, even with a fully embedded TCP/IP, you still need system management and security features that might be best left to a gateway. By pulling some of the networking load into a gateway, you can implement either a proxy or a less-than-complete TCP/IP stack and reduce the processor performance required at each node. For example, a gateway could implement one firewall for several nodes instead of a firewall for each node. Gateways are advantageous in many applications; however, for some devices, such as toys and cameras, users will want to directly connect to their home LANs or ISPs, and anything short of a full TCP/IP won't do.

Note that a direct, peer-to-peer connection needs no gateway. Such a connection might occur when a node dials directly into the server. In such a case, you don't need TCP/IP to pass information back and forth. TCP/IP's advantages are that it connects a device to the Internet, and the device can then indirectly connect to the server. In other words, TCP/IP lets you access a device from any access point on the Internet, not just a specialized server.

WHIPPED CREAM AND CHERRIES

Depending upon your application, you may want more than just a TCP/IP stack. Accelerated Technologies offers the $14,995 Nucleus Net, a TCP/IP stack for the company's $7495 to $12,495, including full source code and no royalties, Nucleus Plus real-time kernel. Together, Net and Plus require 100 to 120 kbytes of program memory and 140 kbytes of RAM, most of which serves as buffers, and can run on a 16-bit processor. Net also supports ARP (Advanced Resolution Protocol), which translates IP addresses to MAC addresses; knowing an IP address is not enough because you need the MAC address to determine the next hop. You might also want a DNS server to convert domain names into IP addresses; a gateway could support the DNS server. If a device has insufficient local storage to hold a hard-IP address, then you'll want to support DHCP to serve temporary IP addresses.

ONE COMPELLING ASPECT OF IMPLEMENTING A WEB SERVER IS THAT IT ADDS AN INTERFACE—THAT COMPUTERS ANYWHERE IN THE WORLD CAN ACCESS—TO DEVICES LACKING INTERFACES.

You might also want to add application support on TCP, such as FTP, Telnet, POP3 and SMTP e-mail, or even SNMP. Some RTOS vendors offer these functions themselves or through third-party vendors.

Perhaps the greatest benefit of embedding TCP/IP is that it allows the embedded devices to support embedded Web servers. Devices don't serve up flashy web pages, although they could. For example, a printer or a SOHO gateway could provide configuration pages or instructions. One primary use of embedded TCP/IP is remote monitoring. For example, a vending machine could dynamically build pages that state how much money it has collected and what kind of snacks are running low, or a meter could report usage statistics. The Web server also enables remote configuration of a device. For example, operators at a central office could change the price of drinks based on the temperature outside; on hot days, when demand for cold drinks is higher, the price increases. Remote upgrading also becomes an option. A device could also update content, such as downloading new advertisements to display. One compelling aspect of implementing a Web server is that it adds an interface—that computers anywhere in the world can access—to devices lacking interfaces. Having an embedded Web server isn't making these functions real; Instead, the Web server plays a role in easily and consistently enabling the functions across a variety of applications and devices.

A device must support both a Web server and e-mail because the Web server serves a passive function in that users must query it for information, whereas e-mail allows the device to actively send either scheduled information dumps or alert messages.

You might also consider device-management software because, as you add devices to a network, management of those devices becomes a major concern. For example, if you have a large network with 1000 devices and want to know their status, it could be time-consuming make 1000 queries. By using a database, a server could aggregate and present information when it becomes available. For example, devices could send or receive calls at night and log status information. Additionally, instead of maintaining a direct socket between the device and the database server, the device could use messaging e-mail to log its information. This feature lets you view the most current data at your leisure. Such a database could also periodically run diagnostic and configuration tasks. Database management could also extend to a gateway that manages a subset of devices. The gateway stores messages from and for the devices in its subnet, aggregates the information, and logs that information to the main database server itself.

Whether to use TCP/IP or a proprietary protocol and bridge depends on your application. In general, TCP/IP requires a 32-bit processor, but some implementations can run efficiently on a 16-bit processor. Also, a relatively high-bandwidth connection is necessary to both prevent TCP/IP overhead from eating all the bandwidth and to justify the cost of implementing TCP/IP. If you send only a few bits, TCP/IP is probably more trouble than value. You need to determine the breakpoint—the difference between using TCP/IP in your design and force-fitting a stack into an application in which it does not fit. Think of what kind of processor, memory, and other resources you need to implement your design and then select the connectivity technology appropriate for those resources.□

You can reach Technical Editor Nicholas Cravotta at 1-510-558-8906, fax 1-510-558-8914, e-mail ednnick@pacbell.net.