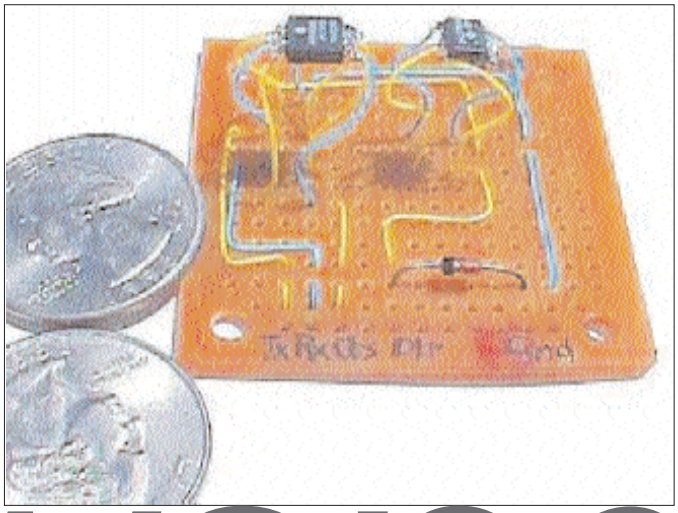
An Indian student in the US has built the world s smallest Web server. He talks to CHIP about his



small wonder

Hariharan Subrahmaniam Shrikumar, an Indian student at the University of Massachusetts, USA, has built the world s smallest Web server that he calls the iPic from a 49-cent chip bought at an electronics store! His invention beat a dozen competing devices to bag the title. The Matchbox Server constructed by Vaughan Pratt, a professor at Stanford, had earlier held the distinction.

Shri, as he is known, says that the potential of the iPic is boundless. The iPic is about the size of a match head . I think it will be very difficult to beat this one, Shri says. It will be interesting to see people try to better it. The iPic can be connected to the Internet through a serial port and does not require a keyboard or display it can be controlled by any machine connected to the Web.

Isn t there more to the iPic than just being the smallest Web server in the world?

Yes, it was much more than just breaking a notional record. A device like the iPic has many practical applications. It has a tiny standards-compliant TCP/IP stack many different applications can be written based on TCP/IP and Web standards, and the possibilities of this are now open. Serving Web pages is only a very small part of the story. Remember that a Web browser not only fetches information it can also be used to make selections, click on switches and check-boxes, enable or disable features, change settings and send them to the remote computer to make the settings take effect. With the iPic, that remote computer has shrunk to the size of a match-head, and costs less than a dollar. This means you can connect practically any device or appliance to a network, and control it over the network.

What kind of practical applications are you referring to?

One good example is this: most appliances today are electronic, and many of them come with small embedded microcontrollers. However, each has its own user interface and, due to cost and size restrictions, the interface often consists of merely a couple of cursor buttons and a minimal LCD/LED display. You have to walk through multilevel menus to change settings, whether you are using a microwave, a coffee machine or even a fax machine or laser printer. Often, this has you running for the user manual.

Use of Web standards can change this. Your PC already has a nice general purpose dynamically configurable GUI the Web-browser that runs Java. The appliance or device could export its control panel as a Java applet, and you can control the device using graphical primitives and Help screens, rather than punch tiny keys with pudgy fingers. Also, we can use inexpensive Web terminals or thin clients, and get rid of the PC altogether.

Some examples?

Devices on such a network can talk to each other and coordinate their settings.

You set a new wake-up time on your alarm clock, and it automatically updates the time setting on your automatic coffee brewer, and also turns on your room heater 30 minutes before you wake up. This is difficult to do without communication standards, and the Web provides a good standard for us to build such appliances on.

The appliances already have the intelligence, and the Internet provides the connectivity. Devices like the iPic connect them together, and allow appliances to be networked at a very affordable cost.

You mean there are huge commercial applications for the iPic ?

The possibilities are quite boundless. In essence, every device and appliance we use today can be connected to an easy-to-use control panel. So far, we had to learn the language of the device and consult the user manual we were

subservient to the appliance. Ideally, the machines should be subservient to us, and speak a language of our choice.

If we want a VCR to work with a TV, we now need to look up the VCR s output channel setting, then pick up the remote for the TV to flick to that channel.

The equipment should talk to each other and get settings from each other as required. We should not need to read settings from one device and feed it to the other. Devices like the iPic can allow appliances to connect to each other, perhaps by sending signals over electric wiring, infrared, wireless or cables and make this dream possible.

What exactly is the TCP/IP stack design and the specific features implemented in the iPic?

The TCP/IP stack is written in a modular style as much as is possible with such space constraints, that is. It consists of a media access module, a protocol processing engine and an application interface. The application interface is not the socket interface, instead, it is described in terms of task precedence relationships in a real-time OS, and communication and interlocks between these tasklets constitute the API interface. The API, therefore, support not only static files, but also dynamically generated data, which you can think of as something akin to cgi-bin.

In the demo server, the media access

“ Devices like the iPC allow various appliances to be networked at a very affordable cost”

mechanism used is SLIP, which operates over a common serial line. This can be replaced by PLIP (parallel line IP) or Ethernet. The innovation is in the TCP/IP firmware core, and this can be used with any of these media.

The iPic can connect to any standard SLIP router. It gets IP packets and replies to them according to the TCP and IP protocols. During a recently concluded technical test period, for two weeks, the iPic was connected to a Linux machine configured as a router.

The router does not do anything other than forward packets to the iPic and send the packets its gets from SLIP on to the Internet. It does not process the packets in any other way.

Why did you use SliRP and not PPP ? Why have you connected your PC with the iPic server and why do they share an IP address?

At this moment, for a variety of practical reasons including the need to get an IP address assigned on our LAN the iPic is connected to a machine in such a manner that it can share its IP address. SLiRP does this as it puts a regular SLIP connection to the iPic and the iPic is doing all the packet processing by itself. SLiRP is exactly like IP masquerade in Linux, only it is done as user process instead of a kernel implementation it is only forwarding packets and mapping IP addresses so that the same IP address can be shared by two machines.

The SLiRP arrangement is not absolutely necessary; the iPic can be directly connected to the LAN or to a router without SliRP or IP masquerade. In fact, one verison of the iPic uses PPP, and connects directly to a modem. It can dial into any ISP, or you can dial into it and connect to it using plain PPP.

Could you give us more details about the file system you ve developed ? How is it implemented?

The file system is very simple. However, it is important to note that it is not hard-coded. There are 64 file slots, and you can think of these as 64 entries in a File Allocation Table (FAT). No subdirectories exist since it is a flat file system why would you need directories when you have only 64 files? Each file can be from 512 bytes to 4 KB in the current implementation, but this can be easily extended to up to 1 MB (or whatever the capacity of your memory device is). The current FAT sizing table entry of 512-4096 bytes is just perfect to allocate the 32 KB I have in the 24lc256 EEPROM into 32 different files of different sizes. You can see all the files in the filesystem at www-ccs.cs.umass.edu/~shri/ iPicTech. html. They add up to 32 KB the capacity of the EEPROM.

The FAT is also stored on the EEP-