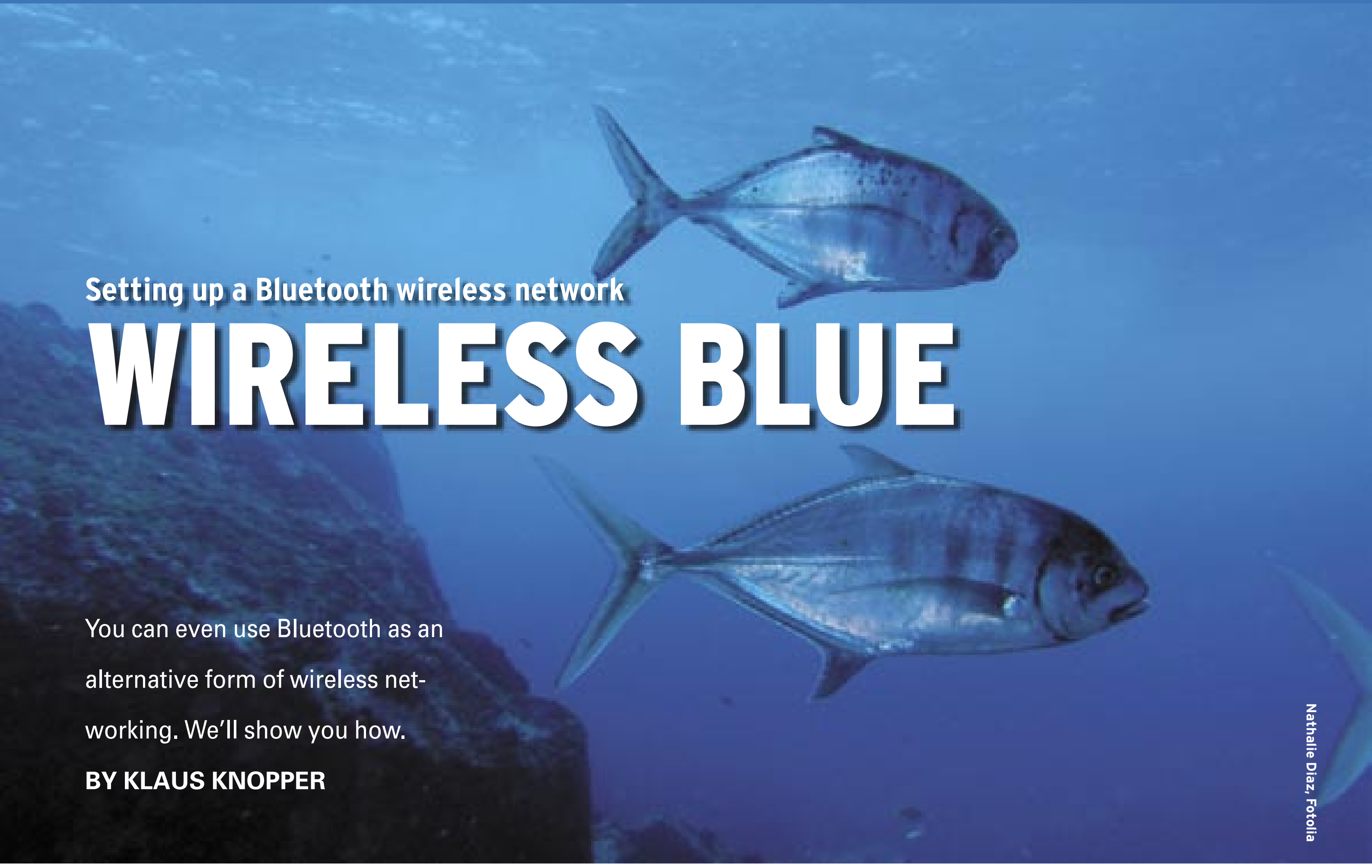

Setting up a Bluetooth wireless network

WIRELESS BLUE

You can even use Bluetooth as an alternative form of wireless networking. We'll show you how.

BY KLAUS KNOPPER

A single Bluetooth adapter supports up to seven simultaneous connections. You can connect a Bluetooth printer, cell phone, and keyboard to a single adapter. You can even use the Bluetooth adapter as an additional wireless network device. As a networking device, Bluetooth delivers a decent throughput of up to 2Mb (for EDR-capable devices – otherwise about 700kb) with a range of up to 100 meters between the access point and client (with no walls in between).

This article describes a technique for creating an access point that allows clients to connect to a local network and access the Internet. This configuration is similar to a Wifi access point, only I'll use Bluetooth instead of a wifi adapter. This configuration also requires

- two or more computers with Bluetooth adapters,
- a Bluetooth-capable kernel on each computer, and
- the Linux Bluetooth bluez-utils package on each computer.

On the computer acting as the access point, you will also need

- a kernel with the network bridging mode feature and the bridge-utils package,
- a dhcp server (preferably ISC dhcpd version 3),
- ipv4 forwarding,
- an Internet connection (if you want Internet access) with masquerading.

This procedure assumes the classic text-based approach to Linux configuration. Keep in mind that most Linux distributions provide some means for setting up network interfaces through their own GUIs. These GUI tools often interfere with configuration files such as the files described in this article.

If you experience unexpected effects like wrong IP addresses or sudden changes in your network configuration, check for KDE or GNOME services that might "automagically" change the Bluetooth settings. Try to configure the GUI to avoid any changes to Bluetooth. On the other hand, stopping all network configuration services will probably also kill the Internet connection on the computer that's supposed to become the access point. Don't despair if your configuration does not work exactly as specified in this article. It is probably not your fault. You may have to tailor the approach to your own environment.

For my Bluetooth network, I will use the personal area network (PAN) specification. The alternative dialup networking (DUN) method is much slower because the protocol has a lot of overhead that is not needed for this configuration. Note that, because I am working at the network setup level, almost all commands in this article must be issued as root.

Setting Up a Bluetooth Access Point

Before you start to set up your computer as a Bluetooth access point, make sure the Bluetooth adapter is working. If the command *hcitool dev* reports something that looks like

```
Devices:
    hci0    00:04:0E:92:0E:6A
```

Debian Note

In Debian, you will have to check */etc/default/bluetooth* for *BLUETOOTH_ENABLED=1*; otherwise, the Bluetooth (or bluez-utils) *init* startscript won't do anything useful. If *PAND_ENABLED* is set to *1* there, a *pand* process might already be running. Either kill it, or modify the *PAND_OPTIONS* before re-starting Bluetooth to match the settings described in this article.

your Bluetooth adapter should be operational. Some adapters, like those from AVM, require special firmware in */usr/lib/hotplug/firmware*. A good hint is to check the output of *dmesg* shortly after the adapter has been plugged in.

The next step is to configure the *hcid.conf* file. The HCI daemon *hcid* needs to know which way to set up adapters, which services to present, and which features to support. For a Bluetooth access point, the HCI configuration section shown in Listing 1 should be present in */etc/bluetooth/hcid.conf*.

The example in Listing 1 uses the shared *PIN* "*1234*" for the pairing procedure on incoming connections. That way, you don't have to set up individual passwords for clients. See the article on Bluetooth security elsewhere in this issue for more on Bluetooth PINs.

For PAN, I need to load some kernel modules. Usually, the Bluetooth startscripts take care of loading the necessary modules, but to make sure:

```
modprobe bnep
```

loads the required module for Bluetooth networking. After you enter:

```
/etc/init.d/bluetooth restart
```

(or *bluez-utils restart*, depending on your distribution), new clients that connect to the Bluetooth access point will get a new *bnep** device: one for each client. (The first client will be visible as *bnep0*, the second as *bnep1*, and so on, and you can set individual addresses and routes for each of these devices.)

Of course, from the viewpoint of the network administrator, unpredictable network device names that each require an IP address are very uncomfortable. Luckily, two mechanisms make the handling of these devices easier.

First, I create a "bridge" device that will join all bnep* interfaces into a single device with only one IP address on the computer acting as an access point. You need bridge-utils for this. I will use a private Class C type network *192.168.192.** in this example:

```
brctl addbr pan0
brctl setfd pan0 0
brctl stp pan0 off
ifconfig pan0 192.168.192.1
```

The *setfd* and *stp* lines are optional optimizations that disable some Ethernet bridging features I won't be using.

To make this address configuration permanent, you could (after successful tests) add the entry shown in Listing 2 to the file called (in Debian) */etc/network/interfaces*.

You are probably wondering where the Bluetooth network device bnep* is actually used. Actually, I have no Bluetooth connections yet, so no bnep* devices have been added to the bridge. Adding the Bluetooth devices on demand is something that has to be done right after a client connects.

Listing 1: HCI Configuration

```
#
# /etc/bluetooth/hcid.conf - HCI daemon configuration file.
#

# HCId options
options {
        # Automatically initialize new devices
        autoinit yes;

        # Security Manager mode
        #   none - Security manager disabled
        #   auto - Use local PIN for incoming connections
        #   user - Always ask user for a PIN
        #
        security auto;

        # Pairing mode
        #   none  - Pairing disabled
        #   multi - Allow pairing with already paired devices
        #   once  - Pair once and deny successive attempts
        pairing multi;

        # Default PIN code for incoming connections
        # Please change this!
        passkey "1234";
}

# Default settings for HCI devices
device {
        # Local device name
        #   %d - device id
        #   %h - host name
        name "%h-%d";

        # Local device class, see "man hcid.conf"
        # This class example matches (almost) everything,
        # including PAN.
        class 0x3e0100;

        # Default packet type
        # pkt_type DH1,DM1,HV1;

        # Inquiry and Page scan
        iscan enable; pscan enable;

        # Default link mode
        #   none   - no specific policy
        #   accept - always accept incoming connections
        #   master - become master on incoming connections,
        #            deny role switch on outgoing connections
        lm accept, master;

        # Default link policy
        #   none    - no specific policy
        #   rswitch - allow role switch
        #   hold    - allow hold mode
        #   sniff   - allow sniff mode
        #   park    - allow park mode
        lp rswitch,hold,sniff,park;
}
```

I must make sure every client is associated with the bridge on the access point server. For *on-connect* tasks like this, the *pand* server (which I did not start yet) offers a mechanism through a script that's called every time a new Bluetooth network device comes up:

```
/etc/bluetooth/pan/dev-up
#!/bin/bash
  ifconfig $1 0.0.0.0
  brctl addif pan0 $1
```

Although the *pand* man page says that */etc/bluetooth/pan/dev-up* is called with the new per-connect *bnep* network device as an argument, in my tests, it turns out that this is not the case in bluez-utils 3.7 under Debian. You need to tell the *pand* server to explicitly start the script instead (see below). The seemingly odd line *ifconfig $1 0.0.0.0* just makes sure the device is functional. (The script uses the IP address of the bridge device *pan0* when sending packets to a different network.)

Now, I am ready to start the real Bluetooth PAN server, *pand*, on the server acting as access point:

```
pand --listen ↩
--role NAP --devup ↩
/etc/bluetooth/pan/dev-up
```

The option *--role NAP* tells *pand* that I run in *Network Access Point* mode, which sets up a network, as shown in Figure 1. Another option would be acting as a *Group Network Controller* (*--role GN*), a kind of Bluetooth network bridge that can add more clients to an existing network or just work as a repeater for client-to-client connections, thus extending the range of operation. But I'll stay with NAP for now.

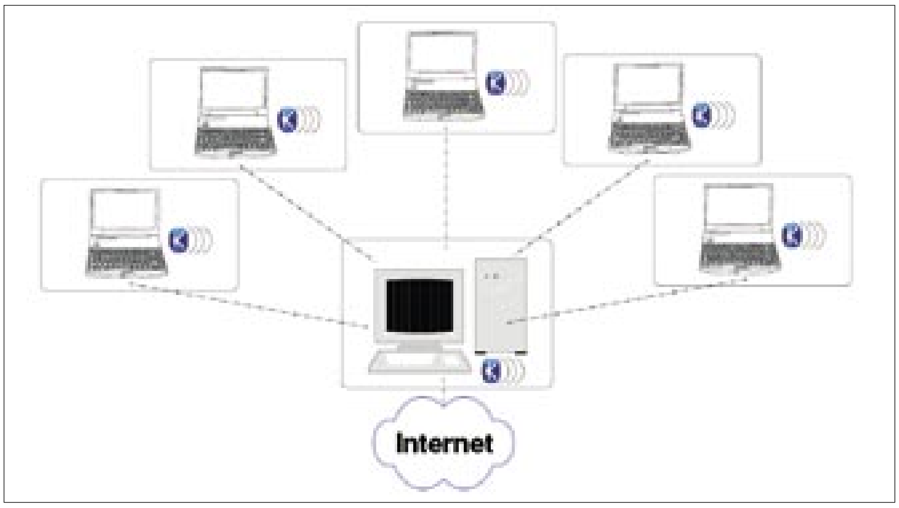

Figure 1: Configure one computer to act as an Internet access point for other devices.

The preceding command will look for a local Bluetooth adapter, start the access point server in listen mode, and fork itself to the background. You can watch its operation in the system log. Alternatively, you can run *pand* in debug mode by adding the *--nodetach* option.

If you have not done this before, enable ipv4 forwarding to share your Internet connection and enable masquerading so that the Internet-side router does not see any private IP networks (Listing 3).

The last *iptables* line in Listing 3 uses the ISDN line(s) as an Internet gateway. Change *ippp+* to the name of the outgoing interface. Of course, Listing 2 is nowhere near a complete firewall; it will let all outbound connections through that are initiated by Bluetooth clients. If you have a direct and permanent Internet connection, you should make sure that either you are not running any vulnerable services on your server or that all incoming connections except the necessary ones are filtered. For some distributions, connection sharing is already enabled for local interfaces, and you won't need some or all of the lines in Listing 3.

An alternative method is to add the external interface to the pan0 bridge with *brctl addif pan0 interfacename*. Unfortunately, this approach offers less control over forwarding between local interfaces. Still, it is worth a try if the forwarding option doesn't work.

Now the setup is complete on the server, and the *pand* NAP service is ready to be contacted by clients. For convenience, I will also set up a DHCP server so clients don't have to set IP parameters manually. Add the section shown in Listing 4 to the DHCP server config file (*/etc/dhcp3/dhcpd.conf* in Debian).

And don't forget to add pan0 to the list of devices that *dhcpd* serves in */etc/default/dhcpd3-server*; just add pan0 to the *INTERFACES* variable there (space-separated). After you enter the command:

```
/etc/init.d/dhcp3-server restart
```

Listing 2: /etc/network/interfaces excerpt

```
[...]

auto pan0
iface pan0 inet manual
        up   echo     "Adding ethernet bridge between LAN and PAN"
        up   brctl    addbr pan0
        up   brctl    setfd pan0 0
        up   brctl    stp pan0 off
        up   ifconfig pan0 192.168.192.1 netmask 255.255.255.0 up
        down echo     "Removing ethernet bridge between LAN and PAN"
        down ifconfig pan0 down
        down brctl    delbr pan0

[...]
```

Starting pand at Bootup

To start *pand* automatically at bootup via the */etc/init.d/bluetooth* init script, check */etc/defaults/bluetooth* for *PAND_ENABLED=1* and *PAND_OPTIONS="--listen --role NAP --devup /etc/bluetooth/pan/dev-up"* (Debian).

Listing 3: Configuring IP Forwarding

```
echo 1 >/proc/sys/net/ipv4/ip_forward
iptables -I FORWARD -i pan0 -j ACCEPT
iptables -I FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -t nat -I POSTROUTING -o ippp+ -j MASQUERADE
```

dhcpd listens on the Bluetooth bridge pan0 for DHCP requests from clients and tells the clients which configuration to use. Of course, you should change at least the name server option to match your network setup. When running your own name server cache, you can just use an IP address that the name server listens on. Otherwise, the setting should match the name server given in */etc/resolv.conf*.

Setting up a Bluetooth Network Client

The setup on the client side is much easier because all the routing and managing is now running on the server. For the first pairing of Bluetooth devices, you might have to start a Bluetooth dbus authentication client, as described in the article on Bluetooth with GPRS:

```
passkey-agent --default ⮒
/usr/local/bin/btpin.sh &
```

/usr/local/bin/btpin.sh is the shell script that enters a pairing PIN on demand.

The Bluetooth server *hcid* will call this script over dbus whenever a PIN is needed to establish a connection. Because an identification key is saved after the initial handshake for devices that have been successfully paired, you usually won't be asked for a PIN when connecting later.

If an application (such as kdebluetooth services in KDE) is already listening on dbus for a pairing pin request, you can skip this command.

Now let the client search for a Bluetooth access point and initialize a connection with

```
pand --role PANU --search ⮒
--service NAP --persist ⮒
--nodetach
```

If you are confident that everything is set up right and you don't want to leave the shell open, you can just omit the *--nodetach* option. However, without it, you will have to check *syslog* for problems because *pand* forks itself into the background immediately and won't show any errors in this shell.

At this point, the client *pand* scans the network for a Bluetooth device that offer a NAP service. Once a device is found, the client *pand* connects to the server *pand*, and both computers get a new bnep device for this connection.

Sometimes, the client does not find the server. This problem can be related to different Bluetooth adapter capabilities, too much wireless noise, or simply bad luck.

In such cases, try to connect directly to the Bluetooth address of the server's adapter. In this example, the adapter address shown on the server by *hcitool dev* was *00:04:0E:92:0E:6A*, so the command line would be:

```
pand --role PANU ⮒
--service NAP -c ⮒
00:04:0E:92:0E:6A ⮒
--persist --nodetach
```

If this command still does not give you a "connected" message, recheck your configuration. Unless you run the server and client in secure mode with an additional *-E* and *-S* option for *pand*, chances are that you are not being asked for a pairing PIN. In general, it is better not to rely on Bluetooth's own encryption and authentication. You should instead use SSH or other encrypted channels for accessing external services.

As soon as you see a bnep0 device on the client, you can request an IP address and default route automatically with

```
pump -i bnep0
```

or a different dhcp client of your choice.

Alternatively, if you do not want to use DHCP, you can set an IP address for bnep0 manually (which should match the network associated with pan0 on the server) and set the default gateway to *192.168.192.1*, which is the pan0 bridge address in this example:

```
ifconfig bnep0 ⮒
192.168.192.100
route add default gw ⮒
192.168.192.1
```

and also set a valid name server in */etc/resolv.conf*.

Adding these commands to the file */etc/bluetooth/pan/dev-up* (and adding a *--devup option* to the *pand* command) saves you the work of setting up the bnep0 IP parameters, so it is sufficient to just start *pand* to get connected.

Listing 4: /etc/dhcp3/dhcpd.conf section

```
[...]

# Bluetooth network/pan0
subnet 192.168.192.0 netmask
255.255.255.0 {
    option domain-name-servers
192.168.192.1;
    option broadcast-address
192.168.192.255;
    option subnet-mask
255.255.255.0;
    option routers
192.168.192.1;
    range 192.168.192.100
192.168.192.200;
}

[...]
```

Conclusion

Many distributions offer GUI configuration for Bluetooth networking. If your system supports the GUI alternative, much of what I did in this article is possible with just a few mouse clicks. Still, it helps to know the underlying technology.

Consider Bluetooth networking when no Wifi access point is in reach or when Wifi simply does not work because of traffic pollution, as is often the case at large expos. ■

INFO

[1] BlueZ project: *http://www.bluez.org/*

[2] Bluetooth information: *http://en.wikipedia.org/wiki/Bluetooth*

[3] PAN HOWTO: *http://bluez.sourceforge.net/contrib/HOWTO-PAN*