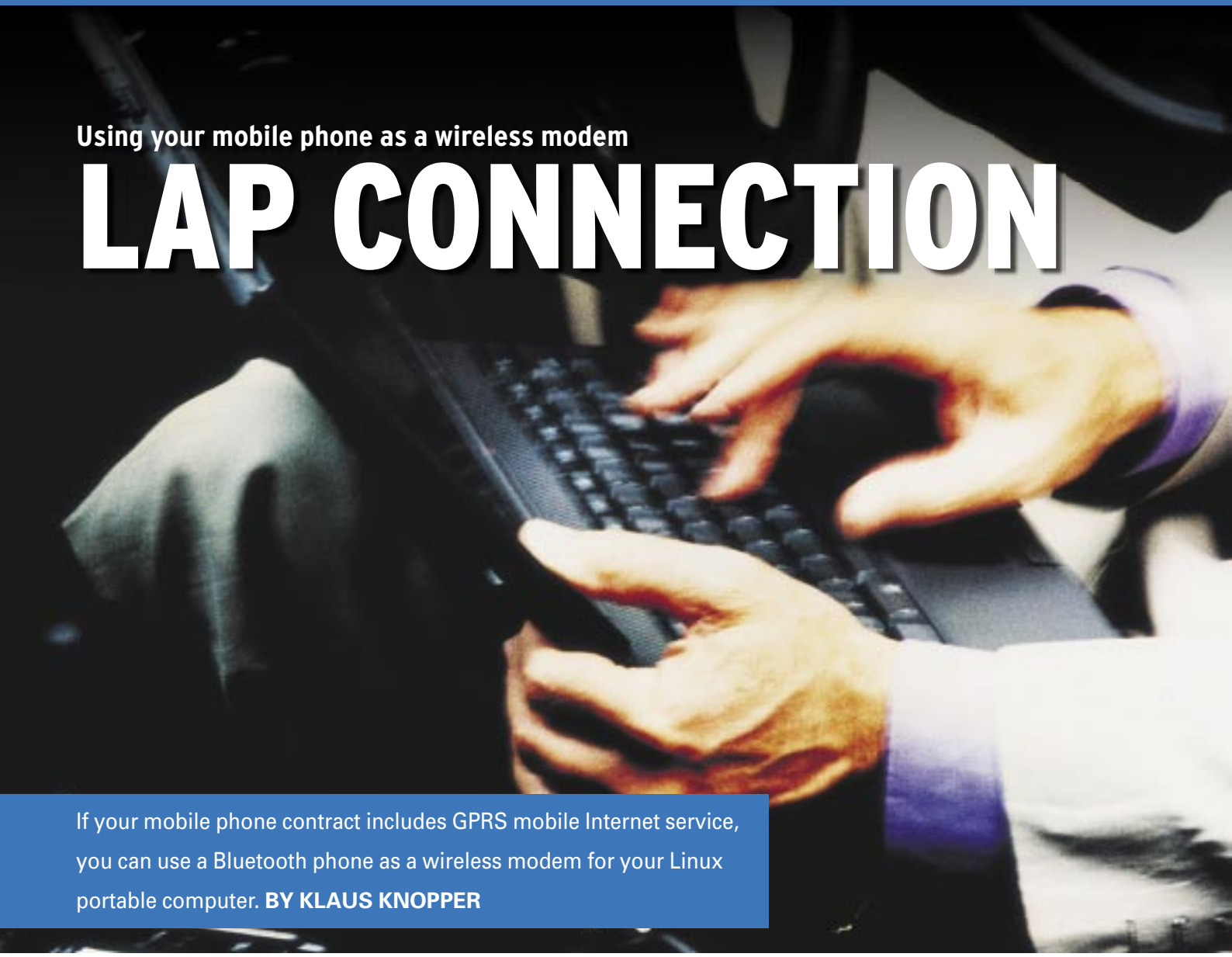

Using your mobile phone as a wireless modem

LAP CONNECTION

If your mobile phone contract includes GPRS mobile Internet service, you can use a Bluetooth phone as a wireless modem for your Linux portable computer. **BY KLAUS KNOPPER**

If you remember the days when computers connected to the Internet through telephone lines, you might be wondering whether it is possible to use an Internet-enabled cellphone or PDA as a wireless Internet connection for a roaming laptop computer. This article describes a technique for connecting a Linux portable computer to the Internet through a Bluetooth connection to a mobile phone. Of course, this procedure requires a mobile phone service plan that comes with some form of Internet connectivity.

The example configuration described in this article assumes Internet access through General Packet Radio Service (GPRS), a mobile networking architecture supported by many wireless providers. A similar approach may also work for other service types. If your service plan provides Internet access through an alternative method, consult your provider for details and adapt the procedures described in this article for your own environment.

Bluetooth Adapter

Before you even start to configure modem emulation for the cellphone connection, you have to make sure the Bluetooth adapter on your computer is working under Linux. The task of bringing up the Bluetooth adapter is fairly uncomplicated with current versions of the Linux kernel, provided all Bluetooth-related kernel options are activated. Make sure you have the bluez-utils package installed, and plug in the adapter. Run the Bluetooth *hci* daemon if it is not already running,

```
sudo ➥
/etc/init.d/bluetooth start
```

Listing 1: dmesg | tail output

```
01 knopper@Koffer:~$ dmesg | tail
02  usb 1-2: configuration #1
    chosen from 1 choice
03  Bluetooth: HCI USB driver ver
    2.9
04  usbcore: registered new
    interface driver hci_usb
05  usb 1-2: USB disconnect,
    address 2
06  usb 1-2: new full speed USB
    device using uhci_hcd and
    address 3
07  usb 1-2: configuration #1
    chosen from 2 choices
08  Bluetooth: BlueFRITZ! USB
    driver ver 1.1
09  bfusb_probe: Firmware request
    failed
10  bfusb: probe of 1-2:1.0
    failed with error -5
```

or if you have an older Debian:

```
sudo ➥
/etc/init.d/bluez-utils start
```

If the Bluetooth daemon is present, the adapter will appear when you enter *hcitool dev*:

```
knopper@Koffer:~$ hcitool dev
Devices:
    hci0    00:03:0D:00:71:8E
```

If this command is successful, just proceed to the next section and get started with configuring modem emulation.

Unfortunately, some Bluetooth adapters, such as the AVM BlueFritz, don't come with preloaded firmware. You will notice this from the output of the command *dmesg | tail* shown in Listing 1.

My preferred procedure is to use a "normal" Bluetooth adapter with firmware inside; however, you can get the AVM Bluetooth adapter to work by just copying the firmware file *bfubase.frm* from the Internet or the driver CD. The following procedure is suggested in the Gentoo AVM adapter HOWTO:

```
root@Koffer# cd ➥
/usr/lib/hotplug/firmware
root@Koffer# wget ➥
ftp://ftp.in-berlin.de/pub➥
/capi4linux/firmware/➥
bluefusb/3-17-53/bfubase.frm
```

After replugging the adapter, *dmesg* should report:

```
Bluetooth: BlueFRITZ! ➥
USB loading firmware
Bluetooth: BlueFRITZ! ➥
USB device ready
```

Modem Emulation

The other part of this Bluetooth GPRS configuration is activating the serial line or *modem emulation* of the Bluetooth stack. Your cellphone now will act very much like a regular modem. For this, you need to create a connection between your Bluetooth adapter and the cellphone, which means you have to define how the connection is made between the adapter and the phone.

You will need to find out which hardware address your cellphone uses. If the cellphone's address is not listed anywhere in the configuration menu of the cellphone, switch on "Bluetooth visibility" for a brief time so you'll be able to scan the Bluetooth environment from your computer with *hcitool*:

```
root@Koffer# hcitool scan ➥
Scanning …  ➥
00:01:E3:07:3E:1A  Blauer Klaus
```

What looks like a MAC address for a LAN adapter is the hardware address you need for setting up a "serial line" to issue modem-like commands.

The text behind the address in the *hcitool* output is actually the *Bluetooth name*, which you set in your cellphone's configuration menu.

After finding out the Bluetooth hardware address, switch to *Enabled with No Broadcast* Bluetooth mode to keep people from bluejacking your cellphone. The *Hidden* mode is sufficient if you know the cellphone's address.

Next, edit */etc/bluetooth/rfcomm.conf* and add this configuration section:

```
rfcomm0 {
 bind yes;
 device 00:01:E3:07:3E:1A;
 channel 1;
 comment "Blauer Klaus";
}
```

Listing 2: /usr/local/bin/btpin.sh

```
01 #!/bin/bash
02
03 PATH="/bin:/sbin:/usr/bin:/
   usr/sbin:/usr/X11R6/bin"
04 export PATH
05
06 TITLE="BLUETOOTH PIN"
07
08 umask 077
09
10 TMP="/tmp/btpin.$$"
11
12 PIN=""
13
14 X="$(/bin/ps --format args
   --no-headers -C X -C XFree86
   -C Xorg 2>/dev/null)"
15
16 XAUTHORITY=""
17 DISPLAY=""
18
19 if [ -n "$X" ]; then
20  DISPLAY=":0"
21  authfile=""
22  for i in $X; do
23   [ -n "$authfile" ] && export
   XAUTHORITY="$i"
24   case "$i" in :*) export
   DISPLAY="$i" ;; -auth)
   authfile="true"; continue ;;
   esac
25   authfile=""
26  done
27 fi
28
29 if [ -n "$DISPLAY" ]; then
30 DIALOG=Xdialog
31 else
32 DIALOG=dialog
33 exec >/dev/console </dev/
   console 2>&1
34 fi
35
36 rm -f "$TMP"
37
38 $DIALOG --title "$TITLE"
   --insecure --passwordbox
   "$TITLE" 8 35 2>"$TMP"
39
40 PIN="$(cat $TMP)"
41 rm -f "$TMP"
42
43 [ -n "$PIN" ] && echo
   "PIN:$PIN" || echo "ERR"
```

Listing 3: /etc/chatscripts/gprs

```
01 TIMEOUT 120
02 ABORT BUSY
03 ABORT "NO CARRIER"
04 ABORT ERROR
05 "" 'ATE1'
06 OK AT+CGDCONT=1,"IP","interne
   t"
07 OK ATD*99***1#
08 CONNECT \d\c
```

Listing 4: /etc/ppp/peers/gprs

```
01 # You usually need this if
   there is no PAP authentication
02 noauth
03 # The chat script for talking
   to the modem
04 connect "/usr/sbin/chat -v -f
   /etc/chatscripts/gprs"
05 # Set up routing to go through
   this PPP link (the ip-up
   script does this now)
06 nodefaultroute
07 # modem port (/dev/ircomm0 for
   IRDA)
08 # /dev/ircomm0
09 /dev/rfcomm0
10 # Speed
11 115200
12 # Ignore carrier detect signal
   from the modem:
13 local
14 modem
15 # Keep modem up even if
   current connection fails
16 persist
17 # Use hardware flow control
   with cable, Bluetooth, and
   USB.
18 crtscts
19 # Be extra verbose
20 debug
21 # Let the server handle all
   configuration
22 passive
23 noipdefault
24 ipcp-accept-local
25 ipcp-accept-remote
26 ipcp-restart 4
27 ipcp-max-configure 20
28 ipcp-max-failure 20
29 lcp-echo-interval 4
30 lcp-echo-failure 0
31 # Special protocol options
32 # asyncmap 0xa0000
33 novj
34 nodeflate
35 nobsdcomp
36 # Tell pppd to set up DNS
   servers set above
37 usepeerdns
38 # Leave username blank.
   Really!
39 user ""
```

If *rfcomm0* is already used by different hardware, you might want to use a different *rfcomm* device number (also shown in the example configuration below).

Instead of restarting Bluetooth now with the command */etc/init.d/bluetooth restart*, you can just add a binding for *rfcomm0*:

```
root@Koffer# rfcomm bind ➥
rfcomm0 00:01:E3:07:3E:1A
```

and check with:

```
root@Koffer# rfcomm
```

to see whether the device is correctly listed as *clean* or *closed* (unless it's already in use).

Shaking Hands

Before you can successfully launch a connection to the cellphone via */dev/rfcomm0*, you have to solve a problem related to the Bluetooth handshake procedure, which requires typing the same Bluetooth PIN or password on both your computer and the cellphone.

Whereas your cellphone will just present a dialog asking for a PIN (which is *not* the secret PIN number that protects your phone from unauthorized phone calls), mechanisms for entering the Bluetooth handshake PIN vary a lot with different GNU/Linux distributions.

In older versions of bluez-utils, a PIN helper program was specified in */etc/bluetooth/hcid.conf* with the *pin_helper* option.

This option is missing from newer versions and is replaced with dbus application-level message exchange. This

UMTS

Universal Mobile Telecommunications Service (UMTS) is an alternative mobile networking technology. Portable computers often connect to UMTS mobile networks with UMTS-ready PCMCIA cards. The UMTS PCMCIA solution is an alternative to the Bluetooth cellphone option in some locations.

What */dev/rfcomm** is for Bluetooth cellphones, */dev/ttyS** is for UMTS PCMCIA cards. The hardware configuration part is again easy: A UMTS PCMCIA card should identify itself as a PCMCIA serial modem, which you can check by looking into the *dmesg* output after inserting the card. Let's assume for now that the card will show up as ttyS2. All mention of */dev/rfcomm0* should be changed to */dev/ttyS2* in the previous GPRS configuration files.

For locally installed cards, no Bluetooth handshake is necessary, but another problem occurs: like many cellphones, the card is usually protected by a personal PIN number that you must enter immediately after switching the device on. In this case, you can submit the PIN with the AT command

```
AT+CPIN=1234
```

which has to be sent to the card's corresponding serial device about four seconds before sending the first configuration or dial commands. The *1234* is the personal PIN number associated with the card (usually printed in the documents that you received when buying the card).

When entering this PIN, which is entirely different from the Bluetooth handshake PIN, instead of sending the PIN manually, you could change the gprs chat-script to something similar to the script in Listing 5, which is based on the eplus UMTS PCMCIA card.

The *1234* has to be replaced by the correct personal PIN, of course.

Please be aware that some cards will only allow up to two or three failed attempts on entering the PIN, at which point the card will be permanently blocked.

The card will be blocked until either a different code is entered to unlock the PIN or the card is sent to the vendor's support service to get unlocked.

The PPP configuration file will have to be changed, at least to use a different serial port (Listing 6).

means it is no longer sufficient to directly use a small dialog/Xdialog-based shell script like the one in Listing 2 for key exchange.

Luckily, it is still possible to use a script similar to Listing 2 with the new dbus-based *passkey-agent*, which is currently located in the kdebluetooth Debian package, even though it is just a console program that can be called with:

```
knopper@Koffer:~➥
$ passkey-agent --default ➥
/usr/local/bin/btpin.sh &
```

PIN on Demand

Because *passkey-agent* does not require root privileges to listen on dbus, you can start *passkey-agent* as a normal user, which makes access to the graphical display easier in the actual PIN input reader. (For instance, you could rewrite *btpin.sh* so it does not have to extract xauth cookies.)

Some Linux distributions might require you to store the Bluetooth PIN for outgoing connections in a file such as */etc/bluetooth/pin*, but I think that providing the PIN on demand is a better approach.

Chat and PPP

Once a modem connection to the cell phone is configured, you can set up a chat script for a GPRS connection (Listing 3). The line *AT+CGDCONT=1,"IP","internet"* is the correct setting for network provider O2; other possibilities could be:

```
eplus:    AT+CGDCONT=➥
1,"IP","internet.eplus.de"
vodafone: AT+CGDCONT=➥
1,"IP","web.vodafone.de"
D1:       AT+CGDCONT=➥
1,"IP","internet.t-d1.de"
```

The line:

```
ATD*99***1#
```

actually initiates the GPRS "dialing" process, which will then require the PPP protocol for local authentication and IP configuration.

A PPP setup similar to the configuration shown Listing 4 should work with most cellphones. Please note that you leave the "user" option argument empty here because authentication against your provider's customer database is really done through your phone number and the cellphone's IMEI device identification. pppd merely handles a reliable connection between your cellphone and your computer on the TCP/IP level, so your cellphone almost looks like a network card to your computer.

Listing 5: /etc/chatscripts/umts-with-pin

```
01 TIMEOUT 120
02 ABORT BUSY
03 ABORT "NO CARRIER"
04 "" 'ATE1'
05 OK AT+CPIN=1234
06 TIMEOUT 4
07 OK-AT+CPIN?-OK AT+CGDCONT=1,"
   IP","internet.eplus.de"
08 TIMEOUT 120
09 ABORT ERROR
10 # use OK AT_OPSYS=3,2 to
   prefer UMTS, but still accept
   GPRS (default setting?)
11 OK ATD*99***1#
12 CONNECT \d\c
```

Listing 6: /etc/ppp/peers/umts

```
01 noauth
02 connect "/usr/sbin/chat -v -f
   /etc/chatscripts/
   umts-with-pin"
03 nodefaultroute
04 /dev/ttyS2
05 115200
06 local
07 modem
08 persist
09 crtscts
10 passive
11 noipdefault
12 ipcp-accept-local
13 ipcp-accept-remote
14 ipcp-restart 4
15 ipcp-max-configure 20
16 ipcp-max-failure 20
17 lcp-echo-interval 4
18 lcp-echo-failure 0
19 novj
20 nodeflate
21 nobsdcomp
22 usepeerdns
23 user ""
```

Getting Connected

After adding the configuration files described in this article and setting the parameters to match your provider and cellphone or UMTS card, you can get connected by starting the PPP daemon:

```
root@Koffer# ➥
pppd nodetach call gprs
```

If you want to see a more verbose description of what's happening, add *debug* right after the *pppd* command.

As soon as two lines with IP addresses appear, you are connected. Don't be confused by the private IP networks that *pppd* displays – these private networks are used internally between your computer and the GPRS provider, and they are masqueraded via real, routed IP addresses. However, because of this design, GPRS and UMTS are currently not usable for running servers that are visible from the Internet, except with a tunnel or proxy.

A Ctrl-C in the preceding example will end the *pppd* command and disconnect your GPRS or UMTS session.

With GPRS and UMTS, as opposed to GSM phone mode, traffic is usually billed by volume (kilobytes transferred) and not by time. But please look up the actual price list to make sure you can afford it. ■

GPRS Hints

Interactive response of GPRS and UMTS is horrible; you will experience up to three seconds of delay between hitting a key and seeing the letter appear in a remote SSH session. You can get used to this, but it is uncomfortable. For reading web pages, you should probably disable autoloading of pictures to save your phone bill and speed up page downloads. Because GPRS is not designed to automatically compress transferred data, you also might experience a dramatic speed (and cost) improvement by using a vtun or SSH tunnel to a remote proxy instead of fetching raw data, especially for text-based protocols like http, imap, or pop3.