people.csail.mit.edu

Communicating with L2CAP

Example 3-4 and Example 3-5 demonstrate the basics of using L2CAP as a transport protocol. As should be fairly obvious, using L2CAP sockets is almost identical to using RFCOMM sockets. The only difference is passing `L2CAP` to the `BluetoothSocket` constructor, and choosing an odd port number between 0x1001 and 0x8FFF instead of 1-30. The default connection settings provide a connection for sending reliably sequenced datagrams up to 672 bytes in size.

Example 3-4. l2cap-server.py

```
import bluetooth

server_sock=bluetooth.BluetoothSocket( bluetooth.L2CAP )

port = 0x1001
server_sock.bind(("",port))
server_sock.listen(1)

client_sock,address = server_sock.accept()
print "Accepted connection from ",address

data = client_sock.recv(1024)
print "received [%s]" % data

client_sock.close()
server_sock.close()
```

Example 3-5. l2cap-client.py

```
import bluetooth

sock=bluetooth.BluetoothSocket(bluetooth.L2CAP)

bd_addr = "01:23:45:67:89:AB"
port = 0x1001
```

```
sock.connect((bd_addr, port))

sock.send("hello!!")

sock.close()
```

As a maximum-length datagram protocol, packets sent on L2CAP connections have an upper size limit. Both devices at the endpoints of a connection maintain an *incoming maximum transmission unit (MTU)*, which specifies the maximum size packet can receive. If both parties adjust their incoming MTU, then it is possible to raise the MTU of the entire connection beyond the 672 byte default up to 65535 bytes. It is also possible, but uncommon, for the two devices to have different MTU values. In PyBluez, the `set_l2cap_mtu` method is used to adjust this value.

```
l2cap_sock = bluetooth.BluetoothSocket( bluetooth.L2CAP )
    .
    . # connect the socket
    .
bluetooth.set_l2cap_mtu( l2cap_sock, 65535 )
```

This method is fairly straightforward, and takes an L2CAP `BluetoothSocket` and a desired MTU as input. The incoming MTU is adjusted for the specified socket, and no other sockets are affected. As with all the other PyBluez methods, a failure is indicated by raising a `BluetoothException`.

Although we expressed reservations about using unreliable L2CAP channels in Section 3.3, there are cases in which an unreliable connection may be desired. Adjusting the reliability semantics of a connection in PyBluez is also a simple task, and can be done with the `set_packet_timeout` method

```
bluetooth.set_packet_timeout( bdaddr, timeout )
```

`set_packet_timeout` takes a Bluetooth address and a timeout, specified in milliseconds, as input and tries to adjust the packet timeout for any L2CAP and RFCOMM connections to that device. The process must have superuser privileges, and there must be an active connection to the specified address. The effects of adjusting this parameter will last as long as any active connections are open, including those which outlive the Python program.