On-Demand TCP :
Transparent peer to peer TCP/IP over IrDA

Jean Tourrilhes, Luiz Magalhaes and Casey Carter

jt@hpl.hp.com
Hewlett Packard Laboratories
1501 Page Mill road, Palo Alto, CA 94304, USA.

magalhae@uiuc.edu, ccarter@uiuc.edu
University of Illinois at Urbana-Champaign
1304 W Springfield Av, Urbana, IL 61801.

This paper describes a novel approach to using TCP/IP applications over IrDA and BlueTooth. First, we look into why so few applications are available over IrDA and what is necessary to make the use of those applications attractive to end-users. Then, we present a new scheme that enable the use of the IrDA communication layer by those applications in a transparent fashion with minimal overhead. We describe the various components necessary to implement such a scheme, IrNET, the name resolver and the discovery manager, and explain how we have implemented those components under Linux. We finish by showing a few examples of use of those components with real applications.

1 Introduction

Recent advances in technology have made it possible to cram computing and networking technology into ever smaller portable devices. As those devices incorporate more data and services, enabling them to participate in ad-hoc networks can be beneficial.

Traditionally, research in ad-hoc networking has focused on the challenges of network autoconfiguration [15] and ad-hoc routing [12]. The results of this work have been implemented for several link layers, most notably Ethernet and wireless LANs. Unfortunately, the techniques developed can't be applied to TCP/IP over IrDA, due to the connection oriented nature of IrDA (see *section 3.1*).

There are at least three well-known methods used to encapsulate TCP/IP over IrDA (see *section 4.1*), all of them require extensive setup or explicit user intervention to establish a TCP/IP network link between nodes, and so can't truly be classified as ad-hoc.

This paper outlines a novel approach to using TCP/IP applications over IrDA. We describe a transparent and seamless method of establishing TCP/IP connections on the fly and without user intervention, allowing the user to ignore the presence of the IrDA layer and its mechanisms.

2 Motivations

While this paper presents results specific to networking and ad-hoc TCP/IP over IrDA, it is part of a more general research project called CoolTown, and therefore its roots lie in research to support ubiquitous computing.

2.1 CoolTown : the user

HP's CoolTown has a web-centric view of environments and their components [1], and aims to bridge the gap between the physical world where the user lives and the computing world where vast amounts of information and services are available.

In CoolTown, spaces, devices and people have web pages, and can interact using HTTP. This intelligent use of web technology allows users to interact with the environment and devices once they have obtained the devices' URLs. A protocol, e-Squirt [27] disseminates these URLs to interested users via infrared. The Web Presence Manager helps the user communicate with various services available to him in the current context and locality [2].

This project is part of CoolTown and is follows the same philosophy. In this case, we are concerned by how the user interacts with wireless networking technology and how we can make this technology useful to him. Usually, wireless connections are used to support mobile hosts. In a ubiquitous computing environment it is desirable that network connectivity should be transparent to the user, if not seamless.

2.2 The benchmark : compact flash

The measure of success of a user interface is its degree of transparency. The more intuitive the interface, the less intrusive and user-friendly it will be. Of course, any interface can become "natural" if it is used sufficiently. Absolute measurements of interface ergonomic are hard, and even comparative measurements tend to be skewed by the process of evaluation and the differing expectations of each interface.

The basic benchmark we have defined to measure the degree of user-friendliness of our wireless interface is the comparison with removable storage. If copying data to a compact flash or floppy and carrying it to the destination (the so called "sneaker-net") is easier than sending it through the network, then the design of the interface is flawed. It is our firm belief that wireless ad-hoc networking can be successful only if it becomes easier to use and more friendly than removable storage. Unfortunately, we are not yet at that stage.

2.3 IrDA : pervasive ad-hoc networking

Support for infrared communication has been present for a long time in laptops, and has seen increased usage as PDAs become more common. Jornadas, Palm Pilots and other PDAs can "beam" information using OBEX over IrDA [8].

When HP started developing its CoolTown project, it made sense to use this widely available (although still under-used) link layer. Its main advantage over competitive technology is its directionality, which allows the user interface to be greatly simplified (just point and shoot, no need for messy on-screen selection). Other benefits of IrDA are its low price, widespread availability, high speed and ad-hoc nature.

The problem with IrDA is that only a handful of specialized applications are available for it.

2.4 TCP/IP is ubiquitous

The IrDA stack offers a fully featured socket API, enabling applications to make rich use of IrDA connectivity. It is fairly trivial to modify and recompile existing TCP/IP applications to make them work over IrDA, except for some UI issues. However, this is not something an end-user can do, and very few application developers have adapted their applications to IrDA.

Our goal is to use any common network application totally unmodified over IrDA, especially the applications the user is familiar with. As the vast majority of existing applications use TCP/IP, this means carrying TCP/IP traffic over the IrDA protocol. The other big advantage of TCP/IP is that it is routable, so it enable various kind of Vertical Handoffs [29].

OBEX [8] is supposed to be the "official" way to build applications over IrDA and BlueTooth, but there is today only a few OBEX application available (most of them only implement push functionality and don't even offer browsing), and OBEX doesn't offer the versatility and large application base of TCP/IP (media streaming, instant messaging, gaming and various java networking applications).

2.5 IrDA and BlueTooth

The current work is based on IrDA, and some parts of the implementation are specific to IrDA. However, the concept is not specific to IrDA, and we plan to extend this work to TCP/IP over BlueTooth [10] when it becomes available.

One limitation of the IrDA stack is that the lower layer, IrLAP, is strictly point-to-point when connected [4]. However, to ensure that our work can be applied to BlueTooth, we use multiple IrDA dongles. Each dongle can carry a single point-to-point link, but the Linux IrDA stack supports multiple dongles simultaneously, enabling us to form point-to-multipoint networks.

3 General overview

Our goal is to transparently relocate the user's favorite applications onto IrDA. As we have discussed, this requires transporting TCP/IP over IrDA, but there are other important issues to consider.

3.1 The main problem : connection setup

The IrDA link layer, like the BlueTooth link layer, is a connection oriented medium [4]. In order for IP packets to flow, we must explicitly create an IrDA connection between two nodes.

All the existing schemes for carrying TCP/IP over IrDA expect the user to explicitly trigger this connection setup. The TCP/IP over IrDA connection will be closed down at the user's request or if the IrDA connection is broken. IrLAN access points are the only exception to this rule, if a device discover one IrLAN Access Point and is already configured for it, the device can automatically connect to it when in range, but this is not an truly ad-hoc scenario.

Since we aim for transparency, we want to eliminate the necessity of user involvement in TCP/IP over IrDA connection establishment.

3.2 Emulating a connectionless broadcast medium

Ethernet is the link layer technology people are most familiar with, and doesn't require this explicit setup. TCP/IP and all the autoconfiguration and ad-hoc mechanism that we want work perfectly on top of Ethernet.

So, we could just pretend that IrDA is just another Ethernet link layer. The idea is to add mechanisms on top of IrDA to make it as similar as possible to Ethernet and hide the specificity of IrDA. In other words, to emulate a connectionless broadcast medium over IrDA.

The techniques are fairly well known, but a bit complex. Whenever a new node is discovered on the IrDA medium, a TCP/IP connection is established to it. Standard IP autoconfiguration techniques are used to configure this new node properly within the ad-hoc network. Optionally, an ad-hoc routing algorithm [12] can be used over this set of point to point links to reduce the number of redundant links so that a device doesn't need to connect at the link layer with every device in range.

The problem is that such techniques have significant costs. First, they are complex : extensive debugging and tuning is needed to make them work well and interoperate properly. In fact, it seems that more work will be needed before those techniques are practical [14].

Second, there is significant management overhead, especially if we want the system to react quickly to the dynamic topology changes common to IrDA systems. In fact, even when the user doesn't need network access, his device will spend much of its energy trying to manage and maintain this ad-hoc network with devices in range, which impacts the CPU and battery life of the device. Using reactive routing protocols such as AODV [13] can only reduce the IP level management traffic, however the link layer connections still need to be maintained.

Imagine a hypothetical user walking down a busy street with a BlueTooth capable mobile phone. Further assume that his phone is establishing a TCP/IP over BlueTooth connection with the cellphone of every other person who passes within range, keeping alive those BlueTooth links and exchanging various routing information. The mobile phone's battery would be flat in short order ; such an ad-hoc network is just not practical on a busy city street.

Finally, there are strict limitations to the number of simultaneous link layer connections. IrDA can have only one active connection per physical port [4]. With BlueTooth, the limit is 7 per radio [10]. Increasing the number of physical ports increases the cost, and is therefore not an attractive option. Since IrDA can connect to only one device at a time, it is vital that we ensure it's the right one.

3.3 On-demand TCP

The type of network traffic on ad-hoc wireless links is usually different from that seen on a wired backbone. Most portable devices tend to be personal clients directly used by

their owner, so traffic tends to be transaction oriented : there is useful traffic only when the user performs a network transaction, otherwise no connection is needed.

Also, the transactions are mostly two kinds, directed (device to device) or toward the infrastructure. Device to device transactions are with people within physical range : the user is physically interacting with someone (i.e. chatting) and wants to complement this with a digital transaction (sending the picture of his new girlfriend or new car). Transactions to the infrastructure use the relevant information or service in the Internet (paying for coffee).

We believe that most ad-hoc interactions will therefore be "one-hop" in nature, from our device directly to a peer or directly to an infrastructure access point. We personally don't want other people to use our bandwidth, CPU and battery life for their private interactions. Note that "our device" might be a set of personal devices configured to work as a unit and linked together by BlueTooth or a similar personal area network technology.

Based on these assumptions, we propose an on-demand TCP/IP scheme, where TCP/IP connections are established only when needed, directly to the intended target, and closed when no longer needed. This slightly increases the complexity of the system and the management overhead, but offers various benefits.

The on-demand scheme proposed act at the IP level (see *section 6.3*), so all protocols part of the TCP/IP stack are handled properly (including TCP, UDP, RTP and ICMP).

3.4 Benefits and constraints

The main advantage to our approach is that the device need not pay the price of setting up and keeping alive an IrDA connection with any peers when it is not communicating. This should result in a tremendous saving of power, since resources are only used when needed.

However, the on-demand nature of our approach precludes the use of the classical IP autoconfiguration techniques. Those techniques usually assume a permanently connected broadcast medium and work on top of IP, whereas by default our approach has no IP connectivity. Therefore, we must use novel autoconfiguration techniques integrating IrDA and TCP/IP.

Also, our solution doesn't solve all usage models and network configurations. One of the most important restrictions is that we can't connect to devices out of range if there is no infrastructure. Some usages and configurations require multi-hop routing and permanent TCP/IP connectivity, and in those specific cases more traditional solutions can be used [12].

3.5 Relation to PPP dial-on-demand

The TCP/IP on-demand scheme that we have described is very similar to dial-on-demand that can be found in most PPP implementations. This scheme deals with the same problem : minimizing the use of a costly resource. When the TCP/IP stack wants to reach the host or gateway on the other side of the PPP link, PPP dials the modem and establishes the link. When there is no traffic on the link, it is disconnected.

The two schemes are similar and use the same underlying mechanisms, but there are significant differences. PPP deals with a single link and single IP address which is preconfigured, although some implementations can use a PPP instance for each of several IP addresses to allow multiple links. On the other hand, our scheme deal with IP addresses with are not known in advance, their number is variable and potentially large, and the number of active links also varies and is usually much smaller than the number of IP addresses, so we need a more scalable solution than what PPP offers.

3.6 The general design

The current design involves four different functional units that must be added to the operating system of the devices we are dealing with (see *fig 3.6*).

The first is IrNET, interfacing the TCP/IP stack on top of the IrDA stack. We assume fully functional TCP/IP and IrDA stacks. This module allows TCP/IP traffic to flow across an IrDA connection.

The second is the IrNET Control Channel. This is a part of IrNET that exports IrDA events to the rest of the system and allows fine control over IrNET (mapping of specific TCP/IP flows to specific IrDA destinations).

The third is the Discovery Manager. It receives events from IrNET and the TCP/IP stack and sets up the system appropriately in response. Intelligence to control the system and establishment of on-demand connections resides in the Discovery Manager.

Fourth is the IrDA name resolver, which performs name resolution over IrDA.

The network applications present on the system don't interact directly with these components, but continue to use the standard system APIs (see *fig. 3.6*).

3.7 Implementation bits and pieces

We have implemented this autoconfiguration scheme for the Linux operating system. The system is operational and has been demonstrated with real applications.

The Linux distribution used is GNU/Linux Debian 2.2 [22], upgraded with kernel 2.4.0 [23]. We tested on HP OmniBook 6000 laptops using the integrated IrDA port (a NSC 87338 FIR chipset) and HP Vectra workstations, using serial IrDA dongles (115 kb/s) or USB IrDA dongles (4 Mb/s).

We mostly reuse existing parts of the Linux OS, such as the TCP/IP stack, the IrDA stack, PPP and the network
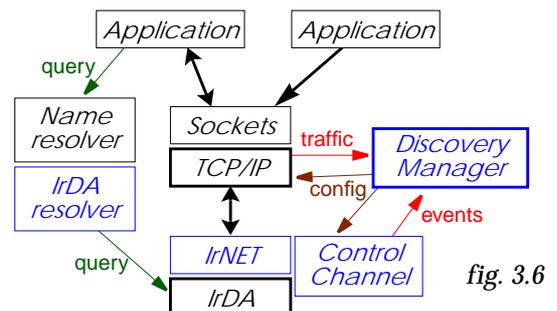

fig. 3.6

applications (web browser, web server, streaming MP3 player).

IrNET and its control channel are implemented in a kernel driver module. The IrDA resolver is a libc module. The Discovery Manager is a regular system daemon.

4 TCP/IP over IrDA : IrNET

The first part of the work is to interface the TCP/IP stack and the IrDA stack. The goal is to encapsulate TCP/IP packets on IrDA connections, and to be able to manage those connections.

4.1 The contenders

There are three different ways to carry TCP/IP over IrDA (see *fig. 4.1*).

The most common is PPP over IrCOMM [7]. This is the method used to communicate with data-enabled mobile phones (those which support IrDA).

IrCOMM is the IrDA stack's simple serial emulation layer, so it's quite straightforward to setup PPP over this pseudo serial port. Unfortunately, this introduces inefficiency due to PPP framing and serial emulation.

The second option is to use IrLAN [6], which is the official IrDA standard for transporting TCP/IP over IrDA, and is implemented in IrDA LAN Access Points. IrLAN is basically an Ethernet emulation over an IrDA socket.

The third option is to use IrNET [9], which is used by Windows 2000 to connect two PCs together (Direct Cable Connection over IrDA). IrNET is synchronous PPP over an IrDA socket, using only the protocol part of PPP and removing both the serial emulation and the PPP framing for greater performance.

4.2 Why we picked IrNET

The PPP protocol has some very nice features, features which are desirable for our project.

The main benefit is that PPP can deal automatically with IP addresses, IP routing and IP configuration through the IPCP negotiation, obviating the necessity of other mechanisms to perform those functions. PPP also has built-in security (authentication [19] and encryption [20]).

PPP is slightly more efficient than Ethernet emulation, as it removes the Ethernet header and can perform IP header compression [17] and IP payload compression [18]. In our tests, the performance of uncompressed IrNET and IrLAN were very close, so it's mainly the rich feature set of PPP that made us prefer IrNET over IrLAN.

The main difference between IrNET and PPP over IrComm is performance, because no PPP framing is done. The other advantage is control, because IrComm doesn't allow to specify the IrDA destination address of the serial connection.

Finally, PPP is usually associated with long term static configuration (dial-up connections). Using PPP in an ad-hoc and dynamic fashion is a challenge that we could not ignore...
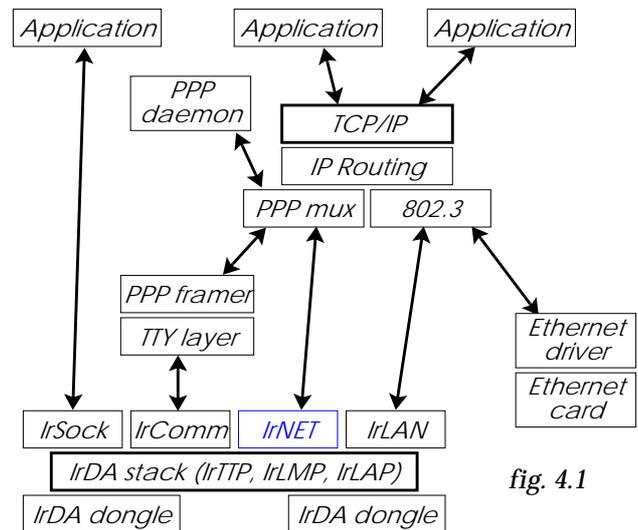

fig. 4.1

4.3 IP autoconfiguration

One of the nicest features of PPP is that it deals with IP configuration for us. The IPCP protocol [16] can negotiate IP addresses for each end of the link and PPP will then perform all necessary network layer setup.

A lot of devices connected to the Internet already have an IP address configured (static or via DHCP on a WAN interface). If no IP address is explicitly given, PPP will use this default IP address of the device for the IrNET connection. This IP address may already be in use by another network interface of the device, but that not a problem. In fact, it's a benefit, because all interfaces of the device will be addressed in the same way.

If the device doesn't have any IP address, PPP will pick a random IP address in one of the non-routable IP subnet (such as 10.0.0.0/24). Those addresses are usually good enough for the kind of short lived directed transactions we are talking about.

PPP assigns two IP addresses at each end of the IrNET connection which may or may not be in the same subnet. However, PPP sets the appropriate host-specific route in the IP routing table, so in practice IP traffic always gets to the correct destination.

PPP can also automatically setup proxy ARP, to enable packet forwarding between the IrNET connection and other network interfaces. Our work does not use this feature of PPP, since we assume that devices are personal and the user doesn't want his resources and battery to be used by other people. On the other hand, proxy ARP could be used in the future to implement an IrNET access point.

4.4 The Linux Implementation

IrNET was not available for Linux, so we implemented it [28]. As Linux offers both a full featured PPP stack and a full featured IrDA stack, it was simply a matter of gluing the two together properly.

The Linux PPP stack is composed of a user space daemon, pppd, and a set of kernel modules. The user space daemon is in charge of the protocol part of PPP. One kernel module is the PPP multiplexer, which interfaces the TCP/IP stack and the

PPP daemon, and deals with common code (such as compression). A second module is one of the framing modules, which performs the link adaptation and usually interfaces with a TTY.

The IrDA stack is a set of kernel modules and composed of the IrLAP, IrLMP and IrTTP protocols. IrSock (the infrared socket API) is built on top of IrTTP.

Although it would be possible to implement IrNET using the standard external APIs as a user space module between the socket and the TTY APIs, for performance reasons, we interfaced IrNET directly to the PPP multiplexer and IrTTP in the kernel. This allows zero-copy communication between TCP/IP and IrDA (if PPP doesn't perform compression), to minimizing latency and reducing the code size of the IrNET module.

The resulting implementation is quite efficient. With a NSC FIR chipset (4 Mb/s link), the TCP throughput measured by netperf [26] is 3.19 Mb/s (uncompressed). The time to setup the IrNET link is less than 800 ms on a 115 Kb/s link (including full IrDA and PPP setup).

4.5 The control channel

Our main contribution to IrNET is the control channel. The control channel is a very simple API (a pseudo file called /dev/irnet) enabling user space applications to interact with the IrNET module in the kernel.

The first function of the control channel is to bind a specific IrNET instance to a specific IrDA destination. When PPP creates a new connection, the IrNET module has no way of determining to which IrDA device the PPP channel should connect. All current TCP/IP over IrDA solutions simply connect to the first device they find. With the control channel, it is possible to specify on the pppd command line the desired destination address, enabling IrNET to properly support multiple devices in range and point-to-multipoint configurations.

The mechanism is simple. The pppd daemon has a command line option (connect) to pass some arbitrary data directly to the input of the PPP driver used. With a regular modem, this is usually the set of AT command to dial the relevant phone number. For IrNET, we have defined a simple set of commands to specify the IrDA address of the connection and a few other parameters.

The second function of the control channel is to export events related to the IrNET connections as well as IrDA discovery events. By reading the /dev/irnet pseudo file, applications are informed when the connection is broken and when new nodes are discovered.

An example event log is :

```
Discovered 8c3478c8 (bougret)
Request from 8c3478c8 (bougret)
Connected to 8c3478c8 (bougret) on ppp0
Disconnected with 8c3478c8 (bougret) on ppp0
Discovered 8c3478c8 (bougret)
Expired 8c3478c8 (bougret)
```

5 The Ad-Hoc Name Resolver

We have chosen IrNET because PPP handles most of the problems related to autoconfiguration. The only thing that PPP doesn't do for us is name resolution.

5.1 The need for name resolution

The TCP/IP protocols provide connectivity, but the only addressing that it knows about is IP addresses. Most people don't want to deal with IP addresses, especially IPv6 addresses, and want to use familiar names.

Various protocols can be used to associate names to IP addresses. The most common is the Dynamic Name Service, DNS [21], designed to work on the connected Internet, where names are organized in a well known hierarchy (my computer is 'bougret.hpl.hp.com'). Other common naming protocols are Network Information Service, NIS (a.k.a. YellowPages) and the use of the static /etc/hosts file.

The user interface of most applications is built around human-readable names. The most common example is HTTP that embeds the DNS name inside the URL. To preserve the user's ability to use names, our system needs to perform name resolution over IrDA.

It may seem a bit paradoxical that we spend so much energy on names when most user interfaces, especially for IrDA, are graphic. But names are present under the graphic skin, and are used between the various component of the system.

5.2 The basic protocol

The classic DNS protocol is too heavy for ad-hoc networking, requires configuration, doesn't handle dynamism, is not peer-to-peer and requires an IP connection (i.e., infrastructure support). To avoid these shortcomings, we implemented a lightweight protocol that is peer-to-peer and doesn't require IP connectivity to resolve names.

The basic idea is to use the underlying link layer discovery. The IrDA stack itself can perform discovery and can build a local database of devices present on the network, including their IrDA address, IrDA nickname and hint bits. Discovery may be done continuously or on demand.

When the user wants want to resolve a destination with a specific IrDA nickname, the resolver query the current IrDA discovery log and extract its IrDA address. This is instantaneous and doesn't generate any additional IrDA traffic.

The second step is to convert the IrDA link layer address into an IP address. We use the IrIAP protocol to query the node associated with the target IrDA address for its IP address.

IrIAP [5] is a basic IrDA protocol that allows IrDA devices to query the IAS database of their peer devices. The IAS is a local database of service record attributes in the IrDA stack. IrIAP can query attributes on an IrDA node by name and obtain their values. The advantage of IrIAP is that it is very efficient, being a link-layer protocol (one request is half the cost of setting up an IrDA socket) and doesn't require server software on the target device.
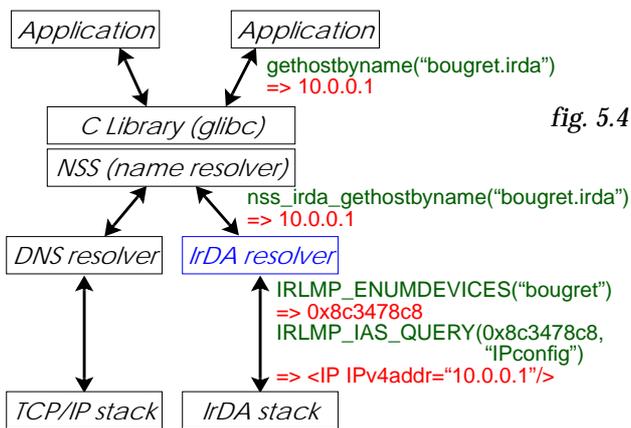
fig. 5.4

Our use of IrIAP is straightforward. The Discovery Manager just add a XML string in the IAS database describing the IP configuration (IP address, DNS name...). Then, we can use IrIAP to query this attributes.

To convert from IrDA address to IP address, the resolver sends an IrIAP query for the attribute named "IPconfig" to the target IrDA device. The result of this query will be a XML string containing the IP address needed.

5.3 The naming convention

In the absence of an infrastructure and a central organizing authority, we can't have a proper name hierarchy. Moreover, we want to resolve IrDA nicknames, so we use a flat name space.

We reuse the "dot" notation of DNS, since most users are familiar with it. Each device on the IrDA link will have a name composed of its IrDA nickname and the suffix ".irda". For example, my computer has the name "bougret.irda".

The name space is not managed, so there may be name collisions, although such occurrences will hopefully be rare on such small networks. One way to resolve collisions is to add an instance number prefix to the name to distinguish different devices (for example "1.bougret.irda" and "2.bougret.irda").

Another option is to use the IrDA attributes found during the operation of the discovery protocol: we can prefix the name with the IrDA class of the device (for example "pda.bougret.irda" and "printer.bougret.irda"). IrDA has only 10 classes of devices and they are used loosely, but this technique will be more useful with BlueTooth, whose discovery protocol SDP is richer and more strict.

We have also introduced an IrDA specific wildcard name, "any.irda", that is not a real device name but resolves to the first device discovered on the IrDA link. This is useful for a line-of-sight link like IrDA because the user can connect to a device without knowing its name by just pointing at it. Of course, the name "any" can be combined with a class prefix (for example "printer.any.irda").

Finally, we can optionally resolve standard DNS names when the hostname part of the DNS name matches the IrDA nickname (as is usually the case). We use the DNS IAS entry of the remote host to make sure we match the proper node.

5.4 The Linux implementation

The name resolver client code is implemented in the C library, as a set of modules with a well defined interface (NSS - Name Service Switch). Those modules include name resolvers for DNS, NIS, host file, and other mechanisms, all configured in the file /etc/nsswitch.conf.

The IrDA resolver is just another resolver module added to NSS (see *fig. 5.4*). It exports to the C library a name resolution handler. It receives name resolution requests, try to perform the resolution and return the result or an error.

Any user space application can query the discovery database of the Linux-IrDA stack through its socket interface. This is done via a getsockopt call. We have added another call to allow user space applications to also perform IAP queries. The IrDA resolver is not specially privileged, it just uses those two calls appropriately.

The current resolver has a few interesting configuration options. We can choose to resolve "any.irda" only if there is a single device in range, or also if there are multiple discoveries (in this case we pick the first one). The resolution of DNS names is optional.

The resolver can also be used to resolve IrDA addresses to IP addresses through the gethostbyaddr() call.

6 The Discovery Manager

IrNET provides the basic method to pass TCP/IP between IrDA nodes, however it doesn't deal with the management and setup of IrNET connections. This is the role of the Discovery Manager.

6.1 On demand TCP/IP

The main goal of the Discovery Manager is to implement on-demand TCP/IP (see *section 3.3*), to create IrNET connections when needed and tear them down when no longer in use. It implements the state machines necessary for this task.
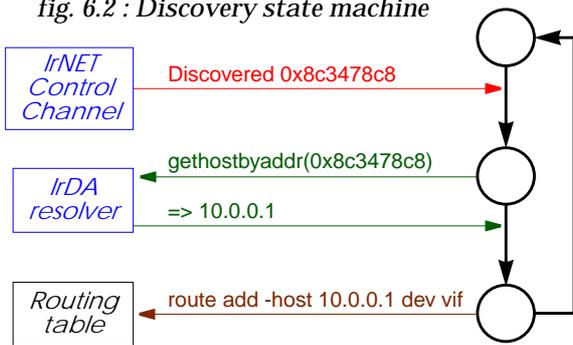
The Discovery Manager needs two types of information to perform its function : it needs to know what are the other peer devices that it can reach through the IrDA link, and when it should establish a link with them.

The whole system is based around TCP/IP and IP addresses, so the first task is to collect IP addresses of IrDA peers, and the second to monitor IP traffic to those nodes.

6.2 Collecting IP addresses

The Discovery Manager uses the IrNET control channel to get discovery events. When a new node supporting IrNET is discovered, an event is generated on the control channel. This event carries the IrDA address of the new node. Upon receiving such an event, the Discovery Manager extracts this IrDA address and, if this IrDA address is not already known, it then uses the IrDA resolver to query its IP address. The use of the IrDA resolver is just a convenience, the Discovery Manager could easily implement the IrIAP protocol itself. The IP address is added to the Discovery Manager's list of active IP addresses, and the binding between the IP address and the IrDA address is stored.

fig. 6.2 : Discovery state machine

IrNET Control Channel — Discovered 0x8c3478c8

IrDA resolver — gethostbyaddr(0x8c3478c8) / => 10.0.0.1

Routing table — route add -host 10.0.0.1 dev vif

The Discovery Manager also reads Expiry events from the control channel, informing it when nodes have left communication range so that it may remove their IP addresses from the active list (the address binding is kept a bit longer in case the node comes back).

6.3 Monitoring IP traffic

The Discovery Manager must monitor outgoing IP traffic towards nodes present in its list of active IP addresses.

Our initial version of the Discovery Manager was using TCP/IP filtering facilities that are present on most standard TCP/IP stacks (such as Packet Filter, Net Filter...). The Discovery Manager was creating a list of IP addresses and receiving events for any packet matching the filter.

However, not all TCP/IP stacks support this functionality, so we changed the Discovery Manager to use a virtual network interface. This is a very simple network driver that offer an Ethernet interface to the stack and exchange data with the Connection Manager (a kernel loopback).
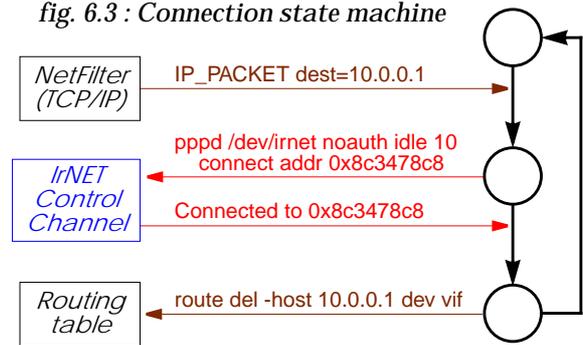
The use of such virtual interface is very simple. The Discovery Manager configure the standard IP routing table to forward all discovered IP addresses on the virtual interface. Then, any packet sent to one of those IP addresses will generate an event containing this packet sent to the Discovery Manager (see *fig. 6.4*).

When the Discovery Manager receives such an event, it first checks the state of the IrDA link associated with this IP address. If the link is unconnected, the Discovery Manager sets up a connection with PPP and IrNET and remove this IP address from the route to the virtual interface. When the IrNET link is up, all packets with this IP address automatically reach their destination. When the link goes down, this IP address is re-routed to the virtual interface.

The Discovery Manager also needs to close unused links. This is done by setting the idle timeout of PPP to 10s so that PPP itself closes the link if inactive. PPP also automatically tears down the PPP connection and the associated IrNET link if the IrDA link is blocked for more than 5s (usually implying that the destination moved out of range). These timeouts may need to be tuned to a specific device or application.

The current system creates a link in response to any packet matching the IP address of the link. This strategy is borrowed from the regular PPP demand mechanism and works well in practice ; unidirectional IP packets are almost always part of a connection and carrying useful information.

fig. 6.3 : Connection state machine

NetFilter (TCP/IP) — IP_PACKET dest=10.0.0.1

IrNET Control Channel — pppd /dev/irnet noauth idle 10 connect addr 0x8c3478c8 / Connected to 0x8c3478c8

Routing table — route del -host 10.0.0.1 dev vif

We do not, however, support broadcast and multicast traffic. Broadcast packets are typically periodic management packets, and do not normally indicate user demand for link connection. Establishing a link with any device in range to exchange such management packets would contradict our goal of preserving power. Multicast packets can be management or multimedia applications ; dealing efficiently with them in such ad-hoc point-to-point environment is still an open research issue and well beyond the scope of this paper.

6.4 The Linux implementation

The Discovery Manager is implemented in Linux as a regular user space daemon and use the various facilities offered by the system.

It reads events on the IrNET control channel by opening the pseudo file exported by the IrNET module (/dev/irnet). It uses the IrDA resolver to translate IrDA to IP addresses.

It performs IrNET connection establishment by launching pppd, the PPP daemon, with the right command line arguments. One of these arguments is the IrDA address that pppd write on the IrNET control channel. The command line can also contain the PPP idle timeout and the IP address of the interface.

The first version Discovery Manager monitors TCP/IP traffic using the Linux NetFilter framework [24] and its associated IPtables library, but it was not optimal.

Currently, the Discovery Manager no longer use NetFilter but a virtual interface. It can use either the Universal TUN driver [25] or the PPP loopback, both standard features of the Linux kernel. The Discovery Manager just route IP addresses on the virtual network interface (either "tun0" or "ppp0") and read packets on a pseudo file (either "/dev/tun" or "/dev/ppp").

The Discovery Manager has been tested two multipoint configuration. The first configuration is when there is multiple peers, each visible behind a separate dongle. In this case, the

fig. 6.4
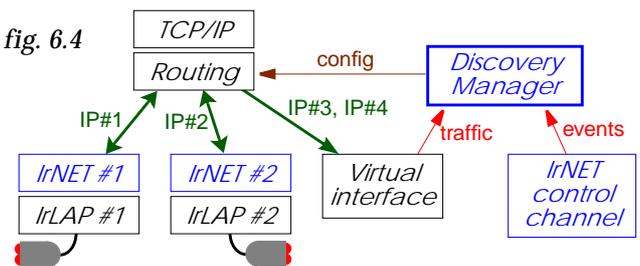
Discovery Manager allow to establish multiple TCP/IP connections to these peers simultaneously.

The second configuration is when all the peers are visible behind the same dongle. In this case, because of the limitations of the IrDA protocol, the Discovery Manager allow to establish TCP/IP connections to these peers only in sequence.

7 Putting all together

Now that we have described the various parts of the system, let's see how they can be used to improve the user experience in a few simple examples.

Of course, it's impossible to list all the potential use of such technology, as it is only an enabling technology that many applications and developers can use for communication over IrDA, and not and end-user application in itself.

7.1 Simple IrDA Web browsing

The HTTP browser has become the user interface of choice for many tasks involving networking, and many appliances contain embedded web servers allowing other devices to browse their content or user interface [1].

If the user wants to browse the content of another IrDA-capable device, he must only point his device toward it, type "any.irda" in the URL field of the browser, and magically the default web page of the other device will appear in his browser. The user can then transparently browse the various web pages of the device.

This is possible using a standard web browser and server, unmodified, on a system that implements our technique. And all web features (post forms, SSL, cookies, multimedia streaming, java plug-ins) are transparently supported.

This is how it works under the hood. As soon as the user's device discovers the other IrDA device, the Discovery Manager puts its IP address in the active list. When the user type "any.irda" in the browser, the browser resolves it and the IrDA resolver returns the target IP address. Then, the browser establishes a connection to the IP address. The Discovery Manager intercepts those IP packets and establishes the IrNET link. After that, the IP packets flow over IrDA and the HTTP server on the other device handle the incoming request.

7.2 Self referenced Squirt

Squirt is a protocol designed as part of the CoolTown project [27], allowing users to remotely control appliances. Squirt push URLs over IrDA using Obex. When the remote appliance receives such an URL, it uses its wired internet connection to fetch the content of URLs and do the appropriate thing with it (display, play or print it for example).

However, this protocol currently applies only for documents available on the Internet. By using our technique, the remote appliance can also transparently fetch documents on the squirt sender. The squirt sender just needs to pass a URL referencing itself (its DNS name) and the local document.

An example is an Internet radio [3] developed as part of the CoolTown project. The user can squirt a URL referencing a local MP3 file, the Internet radio will query and stream the file from the user device over IrDA and play it.

7.3 Network neighborhood

The usual user interface for IrDA is a graphical window showing icons associated with each of the devices discovered on the link. The user can communicate and perform actions on these devices by manipulating their icons.

Using our setup, any network application can be integrated easily into this interface, without changes.

For example, if the user want to add telnet support to this GUI, he can add an item in the contextual menu associated with each device containing "telnet %n.irda" (assuming that "%n" resolve to the IrDA name of the device).

8 Conclusions

The IrDA link layer has some unique characteristics that make it different from usual networking technology : IrDA is peer-to-peer, connection oriented, dynamic, and the link is directional.

These characteristics make the deployment of common network applications based on TCP/IP not transparent to the user : the user must explicitly connect the IrDA stack before using the application. Also, due to the ad-hoc nature of the medium, any network solution must be ad-hoc.

In this paper, we have presented a framework and a set of components that connect the IrDA stack on demand and deal with various details of TCP/IP configuration on such an ad-hoc link. This allows the IrDA link layer to be totally transparent to the application and the user. The main advantage of our approach is its simplicity and the reuse of various existing components to ease its implementation.

The components have been implemented in a real operating system and have been used with various real applications successfully.

Our next steps are to implement a similar on-demand TCP framework on 802.11 and BlueTooth, and to integrate these components in our Ad-hoc Vertical Handoff framework [29].

9 References

[1] CoolTown team. *People, Places, Things: Web Presence for the Real World*. www.cooltown.com

[2] D. Caswell and P. Debaty. *Creating web representations for places*. Proc. Second International Handheld and Ubiquitous Computing Symposium, HUC'2000, Bristol, England, 2000.

[3] V. Krishnan, and G. Chang. *Customized Internet radio*. Proc. Ninth International World Wide Web Conference, Amsterdam, 2000.

[4] IrDA. *Serial Infrared Link Access Protocol (IrLAP)*. www.irda.com

[5] IrDA. *Link Management Protocol*. www.irda.com.

[6] IrDA. *LAN Access Extensions for Link Management Protocol - IrLAN*. www.irda.org

[7] IrDA. *'IrCOMM': Serial and Parallel Port Emulation over IR (Wire Replacement)*. www.irda.org.

[8] IrDA. *IrDA Object Exchange Protocol - IrOBEX*. www.irda.org.

[9] Microsoft. *IrTran-P, IrLPT, and IrDA Networking Support under Windows 2000*. www.microsoft.com.

[10] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen and W. Allen. *BlueTooth: Vision, Goals, and Architecture*. ACM Mobile Computing and Communications review, Vol. 2, No. 4, (October 1998).

[11] IEEE. *IEEE 802.11 : Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*.

[12] S. Corson, J. Macker. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations*. RFC 2501.

[13] C. Perkins and E. Royer. *Ad-Hoc On-Demand Distance Vector (AODV) Routing*. In Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99), pages 90--100, 1999.

[14] Erik Dutkiewicz. *Impact of Transmit Range on Throughput Performance in Mobile Ad Hoc Networks*. In Proceedings of ICC 2001.

[15] S. Thomson and T. Narten. *IPv6 Stateless Address Autoconfiguration*. RFC 1971.

[16] G. McGregor and Merit. *The PPP Internet Protocol Control Protocol (IPCP)*. RFC 1332.

[17] M. Engan, S. Casner and C. Bormann. *IP Header Compression over PPP*. RFC 2509.

[18] D. Rand. *The PPP Compression Control Protocol (CCP)*. RFC 1962.

[19] W. Simpson. *PPP Challenge Handshake Authentication Protocol (CHAP)*. RFC 1994.

[20] G. Meyer. *The PPP Encryption Control Protocol (ECP)*. RFC 1968.

[21] P.V. Mockapetris. *Domain names - implementation and specification*. RFC 1035.

[22] The Debian project. GNU/Linux Debian 2.2. www.debian.org.

[23] Linus Torvalds and others. linux-2.4.0.tar.bz2. www.kernel.org

[24] Rusty Russell. *Linux 2.4 Packet Filtering HOWTO*. http://netfilter.samba.org/unreliable-guides/

[25] Maxim Krasnyansky. *Universal TUN/TAP device driver*. http://vtun.sourceforge.net/tun/

[26] Rick Jones. *NetPerf : a network performance benchmark*. http://www.netperf.org/.

[27] Jean Tourrilhes. *e-Squirt for Linux-IrDA*. http://www.hpl.hp.com/personal/Jean_Tourrilhes/IrDA/squirt.html

[28] Jean Tourrilhes. *IrNET for Linux-IrDA*. http://www.hpl.hp.com/personal/Jean_Tourrilhes/IrDA/IrNET.html

[29] Jean Tourrilhes & Casey Carter. *A-Handoff : A framework for fine grained ad-hoc vertical handoff*. To be published.