# On-Demand BlueTooth: Experience integrating BlueTooth in Connection Diversity

Jean Tourrilhes
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto

HPL-2003-51
March 17th , 2003*

E-mail: jt@hpl.hp.com

wireless,
BlueTooth,
TCP/IP,
BNEP,
inquiry

This paper describes our experiences implementing the Connection Diversity framework over the BlueTooth wireless link layer. The goal of this work is to enable TCP / IP applications to transparently use BlueTooth. First, we describe how Connection Diversity interfaces to the link layer and its requirements. We explain in detail various aspects of the implementation, including the management of connections, discovery, Co-Link and name resolution. This implementation allowed us to test various usages models, and we report their performance characteristics. We then suggest a few improvements to the BlueTooth implementation to improve the suitability of BlueTooth for peer to peer applications.

* Internal Accession Date Only

# On-Demand BlueTooth :
# Experience integrating BlueTooth in Connection Diversity

*Jean Tourrilhes*

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

*This paper describes our experiences implementing the Connection Diversity framework over the BlueTooth wireless link layer. The goal of this work is to enable TCP/IP applications to transparently use BlueTooth. First, we describe how Connection Diversity interfaces to the link layer and its requirements. We explain in detail various aspects of the implementation, including the management of connections, discovery, Co-Link and name resolution. This implementation allowed us to test various usages models, and we report their performance characteristics. We then suggest a few improvements to the BlueTooth implementation to improve the suitability of BlueTooth for peer to peer applications.*

## 1 Introduction

Connection Diversity is a framework enabling the use of Internet applications for peer to peer interactions over any local wireless technology (e.g. : IrDA, 802.11, BlueTooth...).

The BlueTooth wireless technology started as a humble cable replacement [1], but has quickly evolved into the Swiss Army Knife of wireless technologies, allowing all kinds of appliances to be wirelessly connected. BlueTooth is quite different from technologies already supported by Connection Diversity, so integrating BlueTooth helps validating the original design of the Connection Diversity framework.

BlueTooth wasn't primarily designed as peer to peer, but as master-slave [2]. The experience gained from this integration shows various ways BlueTooth can be used for peer to peer applications and illustrates how some design features of the BlueTooth protocol impact user experience.

## 2 Connection Diversity

This section describes the main features of the Connection Diversity framework and its Link Adaptation Layers.

### 2.1 Motivation, assumptions and usage model

Connection Diversity explores how mobile devices can interact using the wide variety of wireless technologies existing today, with a special emphasis on peer-to-peer and ease of use. One goal is to bring the ease of use of wireless technology to the same level as removable storage [10].

A principal underlying assumption of Connection Diversity is wireless diversity : the availability of multiple wireless technologies with different characteristics in each information device. All applications are TCP/IP based to achieve link layer independence, and we want to enable existing popular network applications unmodified.

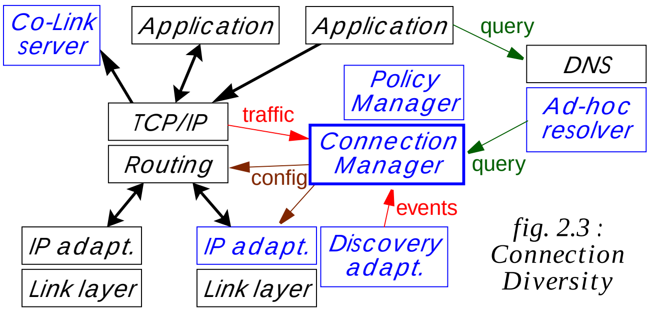

*fig. 2.3 :* Connection Diversity

Connection Diversity offers a simple usage model for peer to peer applications, where a user, through his mobile device, interacts locally with other physically nearby users or appliances in the environment (peer to peer). We don't consider other usage models such as access to infrastructure (wireless Internet) and PAN (master device to peripherals).

### 2.2 On-demand TCP, P-Handoff and Co-Link

The work presented in this paper extends previous work done with On-demand TCP, P-Handoff and Co-Link.

*On-demand TCP* enables peer to peer TCP/IP on a wide variety of wireless links [10]. TCP/IP connections are automatically established and configured over the wireless link when applications need them, between two peer devices, without the need for infrastructure, and then closed down.

*P-Handoff* enables transparent migration of peer to peer TCP connections between wireless links [11]. P-Handoff doesn't require any infrastructure and is fine grained, allowing flexible use of available links. A Policy Manager tries to optimally use those links for each connection based on range, speed and cost.

*Co-Link* enables the use of any wireless link to activate and configure another wireless link [12]. This allows a device to use the most power efficient links for discovery and enable higher performance links only on-demand.

### 2.3 Generic architecture

The Connection Diversity framework is composed of various components inserted in a standard operating system. It currently fully supports IrDA, BlueTooth and 802.11.

The Connection Manager (*fig. 2.3*) is the central controller, a daemon managing the various wireless interfaces of the system and mapping application connections to those [13]. The Connection Manager monitors both peer discovery and outgoing connection requests to implement On-Demand TCP and P-Handoff (*section 2.2*).

### 2.4 Link Adaptation Layer requirements

Different wireless technologies present different APIs, different operating characteristics and topologies [11]. The core and methods of Connection Diversity are generic but require a *Link Adaptation Layer* (LAL) for each wireless technology we want to manage [10]. We enumerate those requirements here.

### 2.4.1 Discovery Management

The Connection Manager depends on the knowledge of which peers can be reached via each link layer, so it can decide what potential connections can be routed on each link layer.

The LAL needs to provide *peer discovery*. Wireless Discovery is not trivial [12], most link layers offer built-in facilities for discovery, and we want to reuse those for efficiency [10]. The LAL needs to know within a reasonable time when a new peer is discovered. It also manages *peer expiry* : it must keep track of discovered peers and remove them from the discovery log when they are no longer reachable (again, within reasonable time).

### 2.4.2 Peer IP Identity

Most often, the link layer discovery only reveals the link layer identity of peers discovered (MAC address). However, both the Connection Manager and the Ad-Hoc resolver need the *IP identity* of those peers, consisting of a Globally Unique IP address and a DNS name [10]. Therefore the LAL needs to convert peer link identities to peer IP identities.

### 2.4.3 IP adaptation

Applications are TCP/IP based, so the LAL needs to transport IP traffic over the link layer. This requires the proper encapsulation of IP packets in link layer packets, and the proper setup of IP configuration and routes.

### 2.4.4 Connection Management

Many link layers are connection oriented and don't offer automatic connection management, leaving it up to the user to connect devices together. The Connection Diversity framework automates this connection management [10].

To enable TCP/IP traffic, the LAL needs to be able to *create link connections* to the desired peers, and to *close* those, based on its routing decisions. It also needs to detect and close *idle* link connections.

Most wireless links are unreliable, so the LAL must monitor those connections for link failures. It needs to know when the link detects likely failure conditions (*blocked* link), and also when the link layer *destroys* the link connection because of this failure condition. We usually prefer to have those two events separate [11], because it's more efficient to monitor the likely failure condition while the link is still connected (even with error) and because disconnecting and reconnecting the link layer incur a large overhead.

### 2.4.5 Ad-Hoc Name Resolver

The name resolver allows to translate both the DNS names and link local names of peers into their Global IP address [10], without using a global infrastructure. A *resolver module* is needed for each link layer, with corresponding *link local names*.

### 2.4.6 Co-Link support

Co-Link uses HTTP requests and is mostly link layer independent [12]. Co-Link needs to query and represent a *link layer configuration*, and must be able to activate the link layer and apply efficiently such configuration.

## 3 The BlueTooth Adaptation Layer

To better understand the integration of BlueTooth into the Connection Diversity framework, we did a complete implementation of its Link Adaptation Layer. We also implemented additional features in the BlueTooth Adaptation Layer to enable additional usage models (*section 5*).

### 3.1 The BlueTooth link layer

BlueTooth is a wireless communication standard initiated in 1997 by Ericsson and Intel [1] and now managed by the BlueTooth SIG (Special Interest Group) [2]. BlueTooth was influenced by the IrDA [5] and USB [6], and offers the functionality of a wireless USB and serial cable replacement.

Like IrDA, and as opposed to 802.11, the BlueTooth link layer is connection oriented, so two BlueTooth devices explicitly need to connect to each other before being able to exchange any data [2].

### 3.2 Discovery Management

BlueTooth offers a link layer discovery process called *Inquiry*. The Inquiry procedure returns the list of *BdAddr* (BlueTooth MAC addresses) of devices that can be reached.

Most devices periodically check if they need to answer Inquiries (Inquiry Scan mode - every 1.28s for 11ms). A device performs an Inquiry by repetitively sending requests and collecting answers from its neighbours (*fig. 3.2*).

The default Inquiry duration is 12s. Due to the design of Inquiry Scan mode (delayed answer [2]), the minimum time to get any answer from Inquiry is 4s, and the probability to get an answer from a peer within 4s is often below 50%.

We have implemented a *discovery manager* that can use Inquiry to build a discovery log (*fig. 3.5*). It performs a periodic Inquiry for 4s every 60s (*table 1*). This aims to tradeoff the latency of discovery, the length of time the interface is unusable (while doing Inquiry) and the Inquiry overhead (both in throughput loss and power - *section 6.1*).

Once a peer is discovered, we need to keep track of it and manage its expiry. This is done through the periodic Inquiry ;

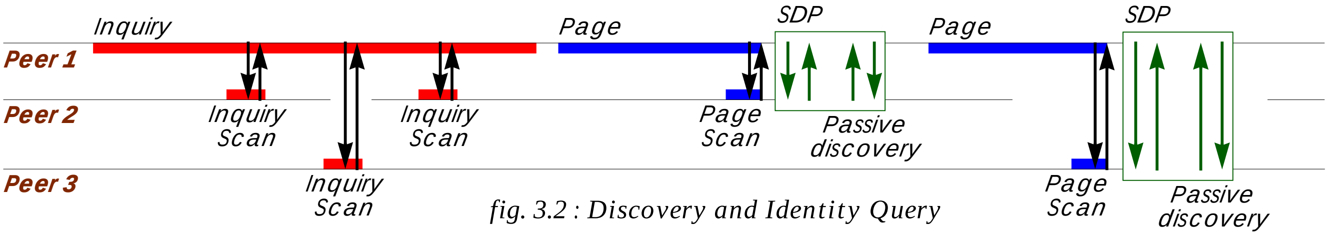

fig. 3.2 : Discovery and Identity Query

if a peer is not discovered for 10 successive Inquiries, it is expired and removed from the discovery log (*table 1*).

Inquiry in the discovery manager can be turned off, and can also be triggered on-demand by the name resolver. The additional “auto” mode allows tracking of discovered peers : Inquiry is off by default and enabled only when there is a valid peer in the discovery log.

The discovery manager can also perform *passive discovery*, i.e., to discover peers without doing any Inquiry. Peers performing Inquiry don’t reveal anything about their identity. However, whenever a peer connects to us, we can be notified of it and get its BdAddr. The discovery manager monitors this event and adds the BdAddr of every incoming connection in the discovery log (if it doesn’t already exist).

The discovery manager also monitors Inquiries triggered by other applications on the device and collects their results.

***Table 1: BlueTooth parameter settings***

| ***Parameter*** | *standard* | *new value* |
|---|---|---|
| Periodic Inquiry period | - | ~60 s |
| Periodic Inquiry duration | - | 3.84 s |
| Discovery log expiry | - | 10 min |
| On-demand Inquiry duration | 12.8 s | 6.4 s |
| SDP retries | - | 3 |
| Page timeout | 5.12 s | 4 s |
| Page Scan period | 1.28 s | 0.64 s |
| Link Supervision Timeout | 20 s | 5 s |
| QoS latency variation limit | - | 250 ms |
| Connection idle timeout | - | 10 s |

### 3.3 Peer IP Identity

Inquiry and passive discovery only return the *BdAddr* of peers and their *class of device* bit-field, and do not contain any other data that could be used to identify the peer. Therefore, the discovery manager needs to query individually each BdAddr found for its peer IP identity (*section 2.4.2*).

This is done using *SDP* (Service Discovery Protocol). SDP associates metadata to each BlueTooth socket, enabling discovery of their functionality and attributes [2]. The SDP server on each device maintains a list of SDP service records, and any peer can query those records with a simple protocol.

We simply added an additional SDP attribute to the SDP record of the BNEP socket (*section 3.4*). This attribute contains the IP identity of the device (*section 2.4.2*).

Each time the discovery manager finds a new BdAddr, it creates a BlueTooth connection to this peer and fetches the SDP attribute containing the IP identity (*fig. 3.2*).

The BlueTooth connection is based on a *Paging* handshake, and requires the BdAddr of the peer. After Paging completes, the higher level of BlueTooth stack can connect (SDP in our case). The time to perform the SDP request itself is usually small with respect to Page time (*table 2*).

Paging is similar to Inquiry and synchronises the two BlueTooth devices on the same Frequency Hopping pattern [2]. The target device periodically checks if it needs to answer Pages (Page Scan mode - for 11ms every 1.28s). The initiator sends Page requests until it gets an answer or timeout.

To minimise connection latency, we reduced the Page timeout from 5s to 4s and the Page Scan period from 1.28s to 0.6s (*table 1*). The peer itself may be doing an Inquiry and unable to answer us, so we will retry the SDP query up to 3 times (once after each successful Inquiry or passive discovery) before marking the discovery log entry invalid.

Performing a SDP query on each peer is time consuming (*fig. 3.2*), therefore to improve scalability the peer identity is cached in the discovery log. Since not all peers answer SDP requests, especially those that don’t support Connection Diversity, we pre-filter peers based on their *class of device*.

### 3.4 IP adaptation

We decided to use a simple subset of *PAN* to do IP adaptation. PAN (Personal Area Network) [3] is one of the standardised networking profiles of BlueTooth, designed specifically for creating ad-hoc networks of devices or connecting to dedicated access points.

The subset of PAN we use is *BNEP* (BlueTooth Network Encapsulation Protocol), which is a direct encapsulation of Ethernet frames over a BlueTooth L2CAP socket (*fig. 3.5*).

The IP address configured at each end of the link is the node Global IP address [10], so there is no need for dynamic IP configuration. The BlueTooth Manager also sets up a host IP route and a ARP proxy entry for each BNEP connection, so that packets are properly routed.

### 3.5 Connection Management

When the Connection Manager requests a link connection, the BlueTooth manager creates and configures it. It first connects to the peer using Paging (*section 3.3*), then creates a BNEP connection using the standard BNEP API, finally configuring IP and the route. The BlueTooth Manager maps the peer connections to the various BlueTooth interfaces available and attempts to find the best BlueTooth interface for each one. It enforces the 7 slaves and 1 master limitation [2], and has basic master/slave switch support [2].

There is no facility in BlueTooth to detect idle links, so we use Netfilter and optional KeepAlive packets to monitor IP traffic. Netfilter [9] is the standard packet monitoring facility of the Linux kernel and allows the BlueTooth Manager to count incoming and outgoing packets on the BNEP connection. After 10s without seeing any activity between two peers, the BlueTooth Manager closes the associated connection and puts the peer back in demand mode (*table 1*).

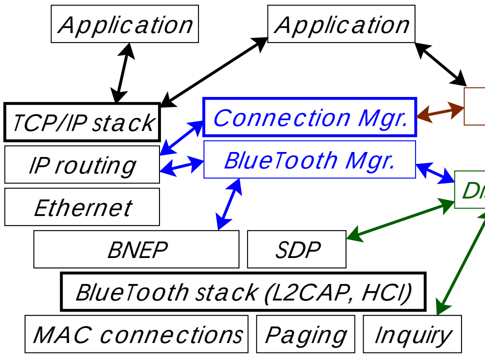

*fig. 3.5 BlueTooth Adaptation Layer*

To detect loss of connectivity, we use the underlying BlueTooth facility : the *Link Supervision Timeout* dictates the time a BlueTooth link remains alive without an answer from the peer [2]. We set it to 5s (the smallest value larger than an Inquiry, to avoid false positives). When the Link Supervision Timeout expires, the BNEP channel is automatically destroyed, the Connection Manager gets notified of it and usually triggers P-Handoff [11].

We attempted to implement the blocked link event (*section 2.4.4*). The BlueTooth manager sets a latency variation limit of 250 ms in the link layer for each connection. Unfortunately, BlueTooth hardware currently available doesn't generate any QoS events, so we could not test this feature and determine its proper setting (*section 6.4*).

### 3.6 Ad-Hoc Name Resolver

The BlueTooth name resolver module interfaces to both the Connection Manager and the discovery manager (*fig. 3.5*).

If periodic Inquiry is active, the Connection Manager already knows about all BlueTooth peers, and the name resolver only needs to query the Connection Manager cache.

If periodic Inquiry is not active, the cache is empty. In the case of DNS names, the name resolver will return "not found" to avoid impacting the performance of regular DNS queries. The resolver still tries to resolve BlueTooth link local names, because those can be resolved only on the BlueTooth link.

The first form of link local name is composed with the name of the peer and the *.bt* suffix, such as *name.bt*. After the cache lookup, the resolver can trigger an complete Inquiry (via the discovery manager - including associated SDP requests) and wait for the result.

The second form of link local name is composed with the BdAddr of the peer and the *.bdaddr* suffix. After the cache lookup, the resolver can trigger a SDP request on this BdAddr (via the discovery manager).

Unlike the IrDA resolver [10], the BlueTooth resolver doesn't yet support service attributes in link local names. We have found that some current BlueTooth implementations don't set properly the *class of device* bit-field [2].

### 3.7 Co-Link support

The Co-Link configuration [12] data for BlueTooth only contains the BdAddr (BlueTooth MAC address) of the peer. We can not add BlueTooth clock offset, because it is relative to the adapter local clock. The XML fragment looks like :

```
<BT BdAddr="BD:AD:D8:01:23:45"/>
```

When Co-Link activation of BlueTooth is requested, the BlueTooth manager switches on the best BlueTooth interface, extracts the BdAddr from the XML, and passes it to the discovery manager. The discovery manager then directly issues an SDP request on this BdAddr to verify its reachability and get its IP identity. Once the identity is known, the Connection Manager can reroute traffic to this peer.

## 4 Implementation details

Connection Diversity has been implemented on Linux [7]. The hardware used is 3Com USB BlueTooth dongles (CSR chipset, BlueTooth 1.1 compliant, 100m range). The BlueTooth Linux stack is BlueZ 2.3 [8], with its standard SDP and BNEP support.

Both the discovery and identity process are implemented in a standalone daemon. The IP adaptation is the BNEP kernel module of BlueZ. The BlueTooth manager is implemented in a module of the Connection Manager daemon. The BlueTooth resolver is a NSS library [10].

## 5 Usage models and findings

The current implementation of Connection Diversity over BlueTooth is quite flexible and enables various usage models. However, each usage model exposes some usability issues of BlueTooth that would apply to other peer to peer applications.

### 5.1 Transparent usage model

Connection Diversity aims for full transparency : the user and the application should not be aware of the BlueTooth link. In this usage model, we want to support any IP application over BlueTooth without explicit setup. This is also the usage model most compatible with P-Handoff : the IP traffic may be transparently migrated on and off the BlueTooth link at any time based on the policy and link layer events.

The way to achieve this is to set the discovery module to do periodic Inquiry and collect identity of reachable peers. When the Connection Manager detects an application that wants to communicate this peer, it automatically establishes the relevant BlueTooth connection (*section 3.5*).

In this model, name resolution is instantaneous, because all peer identities are cached. The establishment of the link is fairly fast (see *table 2*), because the MAC address is already known (so this time is mostly equal to the Paging time).

The main issue is that each peer has to do periodic Inquiry, which is slow (minutes) and results in a significant number of connection failures (*section 6.1*).

***Table 2: Connection Diversity typical times***

| *Action* | *Typical time* |
|---|---|
| Page (no failure) | 150 ms - 700 ms |
| SDP request (excluding Page) | < 40 ms |
| BNEP + IP setup (excluding Page) | ~ 70 ms |
| TCP connection, transparent mode | 250 ms - 850 ms |
| TCP connection, on-demand mode | ~ 8.5 s (1 peer) |
| TCP connection, Co-Link on IrDA | ~ 2s |

### 5.2 On-demand usage model

The typical usage model for most BlueTooth applications is to have discovery and connection explicitly triggered. Our current implementation allows to reproduce this usage model with TCP/IP applications unmodified.

To enable this, the user must specify in the application only BlueTooth link local names (and not IP addresses or DNS names). Those names force on-demand name resolution, therefore we don't need to run periodic Inquiry.

The link local name specified is resolved by the BlueTooth ad-hoc resolver. As periodic Inquiry is disabled, it triggers a full Inquiry and waits until the discovery module has queried all the discovered peers via SDP. The name resolution process takes a minimum of 7s (see *table 1*) and increases with the number of discovered peers (and this time also depends on the success or failures of the SDP queries).

The BlueTooth destination must also learn the identity of the initiator of the connection, to set up IP properly. When the initiator does its SDP query on the destination, the destination uses the passive discovery mechanism (*section 3.2*) to query back the IP identity of the initiator.

When the name resolution is done, the application starts sending data to the destination. The demand mechanism of the Connection Manager triggers the establishment of the BNEP connection, similar to the previous usage model.

The main advantage of this usage model is that there is no periodic Inquiry, so power consumption is lower and connection setup is more reliable. Unfortunately the whole setup is so slow that it is noticeable to most users (*table 2*). In addition the restriction to only use local link names prevent compatibility with the P-Handoff protocol.

### 5.3 Co-Link usage model

One of the main issues with BlueTooth is the need to perform Inquiry (*section 6.1*). By using Co-Link, we can use a link offering a better discovery process to enable BlueTooth and bypass Inquiry entirely.

The two alternatives that we currently support are IrDA and 802.11. Using 802.11 is problematic because it needs to be preconfigured (ESSID and mode setting). On the other hand, IrDA is a good discovery link [12].

IrDA discovery is relatively low power, efficient and fast. The default setup on IrDA is to have periodic discovery every 3s [10]. The full connection setup (including TCP/IP) over IrDA is less than 1s [11].

The usage model is transparent, identical to our initial usage model (*section 5.1*) with the restriction that the IrDA ports must be aligned. The application can use an IP address, DNS name, link local name or wildcard such as *any.irda*.

After the initial setup over IrDA, the application can start to communicate. In parallel, Co-Link does the HTTP query, enables the BlueTooth port, and does a SDP query to the peer. After those steps are completed, the connection may be migrated to BlueTooth using P-Handoff.

This is a typical run using a SIR link (115 kb/s) :

```
time          event             => action
23:19:33.678  packet on demand channel => connect on IrDA
23:19:34.375  connected on IrDA => forward packets on IrDA
23:19:34.378  packets forwarded => Start Co-Link query
23:19:34.521  Co-Link reply     => connect on BlueTooth
23:19:35.287  connected on BlueTooth, P-Handoff done
```

Another scenario is to use BlueTooth to activate and configure a 802.11 link, in this case the usage model is similar to the two previous ones, and with similar restrictions.

## 6 BlueTooth issues and improvements

This experiment has uncovered some issues with the current BlueTooth implementation and specification that would likely apply to other peer to peer applications. We also present a few simple techniques that would make BlueTooth more friendly for such peer to peer applications.

The BlueTooth specification was designed to be mostly master-slave [2], and by using it in peer to peer mode, we seems to be pushing some of its limits. The peer to peer usage model increases concurrency, two nodes are more likely to do incompatible activities at the same time.

### 6.1 Issues with Inquiry

The single most problematic aspect of BlueTooth is the slow, exclusive and expensive Inquiry procedure.

While performing an Inquiry the BlueTooth interface of a node can’t be used for anything else for its whole duration (such as servicing existing connections or accepting new incoming connections). If a peer tries to connect to it (Paging), it will fail. If two nodes perform Inquiry at the same time, they won’t discover each other. We see those failures fairly often in the discovery process (periodic Inquiry + SDP).

When using periodic Inquiry, it usually takes minutes to discover new peers and expire them (*section 3.2*). The Inquiry consumes significantly more power than other BlueTooth modes. Another issue is that, once connected, the node that is the slave usually loses its ability to perform Inquiry, so can’t keep track of its reachable peers until it disconnects.

The cause of this is both the nature of Frequency Hopping, which requires peer synchronisation, and design choices. In BlueTooth, the node can synchronise to a peer only when this peer goes in Inquiry Scan mode, and to preserve throughput this happen infrequently. The node doing Inquiry also has to transmit in every possible transmission slot [2].

Beyond our current setting of periodic Inquiry (*table 1*), there is not much that can be done to fix Inquiry, because it is a core feature of the BlueTooth specification. The only workaround we can currently think of is to use Co-Link to bypass entirely the Inquiry process (*section 5.3*).

### 6.2 Issues with Paging

If a node tries to connect (Paging) to a node that does Inquiry, it will fail (*section 6.1*). Similarly, if two nodes Page each other at the same time, they both will fail. Therefore, we had to make the Co-Link process over BlueTooth explicitly asymmetric : only the initiator attempts to do a SDP query.

We also had stability problems with the hardware (lockups). When doing passive discovery, we have to wait until the incoming connection is accepted before performing Paging. With Co-Link, we also needed a 20ms delay between the activation of the BlueTooth interface and Paging.

### 6.3 Power saving modes

Many proposals in the PAN working groups make use of BlueTooth power saving modes (*Park* mode) to improve scalability or enable scatternets.

The Connection Manager only establishes link connection as needed and close them down when unused, so we don’t need scatternet and are already saving power. One scenario is to use *Park* mode to improve the discovery process, by keeping track of discovered peers : the node would automatically connect to all discovered peers, put them in Park mode, and periodically poll them.

Using park mode forces the use of a networking model and introduces a significant complexity : we would have to manage a mesh of peer-to-peer connections. Between each pair of nodes, one must be master and the other slave. Some nodes may be parked by multiple masters, and some nodes might be both master and slave (with respect to different peers). The master also will need to periodically unpark each slave to verify if it is still reachable.

The performance of Park mode is not much better than our current solution (using Paging). To wake up a peer, the device has to wait for the park beacon [2], and this is roughly in the same order of time as the Page Scan period.

Finally, park mode does not allow us to eliminate the need for periodic Inquiry. Only Inquiry allows to discover *new* nodes coming into range. As we still need periodic Inquiry to happen, the advantage of using Park mode is marginal, and we believe that the complexity and management overhead of such setup is not justified for our usage model (*section 2.1*).

### 6.4 QoS implementation (Link monitoring)

The Connection Manager needs an event to detect quickly potential link failures prior to the actual closure of the link connection (blocked link - *section 2.4.4*).

The Link Supervision Timeout mechanism closes the link, so can't be used. BlueTooth offers some RSSI and link quality measurements [2], but those are tricky to translate into link failure (need to define threshold and window, if receive traffic stops measurements are not updated) and must be polled (increasing I/O overhead and time granularity).

The only natural way to implement such an event in BlueTooth would be through the QoS mechanism : setting a latency variation constraint in the link layer, forcing an event for transmitted packets delayed by more than this constraint. Delays in transmission are mostly due to retransmissions (and therefore excessive range or interference). Unfortunately, current implementations don't support QoS yet.

### 6.5 Paging probes (Expiry)

The Connection Manager needs a way to keep track of peers it has discovered, and to expire them (*section 2.4.1*).

Currently, this is implemented via the periodic Inquiry. We don't want to use any of the Power Saving mode due to the complexity and lack of benefits (*section 6.3*).

Another solution is to use *Paging probes* [14]. Every BlueTooth node has a known Paging Scan behavior (typically a 11ms window every 1.28s). Once the initial discovery of a peer is done, the node could remember its peer's Paging Scan parameters. Then, it only needs to send a Page at the time it knows the peer is doing Page Scan to verify that the peer is still reachable (and timeout or disconnect immediately).

If the number of peers is relatively limited, this technique would be much more efficient than periodic Inquiry or Park mode. Unfortunately, due to the timing accuracy needed, this can only be implemented in the BlueTooth hardware module.

## 7 Conclusion

The Connection Diversity framework is flexible enough to accommodate the BlueTooth technology. Various modules need to be added to the framework, to handle the Inquiry process, SDP queries and BNEP connections. The techniques we implemented and our configuration of BlueTooth is mostly generic and should apply to other applications.

The current implementation of Connection Diversity can make full use of BlueTooth and offers several useful usage models for peer to peer networking. Unfortunately, most of those models can't workaround the slow and expensive Inquiry process needed to discover new peers.

Based on this experience, we make suggestions of improvements to the BlueTooth implementations to aid peer to peer applications, such as adding QoS support, Paging Probes and using Co-Link.

## 8 References

[1] Anders Edlund and al. *MC-Link*. Rev PA2, 15.10.97.

[2] Bluetooth SIG. *Specification of the Bluetooth System*. v1.0b. http://www.bluetooth.org.

[3] Bluetooth SIG. *Bluetooth Network Encapsulation Protocol (BNEP) Specification*. http://bluetooth.org.

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

[5] Patrick J. Megowan, David W. Suvak & Charles D. Knutson. *IrDA Infrared Communications: An Overview*. http://www.irda.org.

[6] USB-IF. *Universal Serial Bus specification v1.1*. http://www.usb.org.

[7] Linus Torvalds and others. linux-2.4.0.tar.bz2. http://www.kernel.org

[8] Maksim Krasnyanskiy and al. *Linux BlueZ Howto*. http://bluez.sourceforge.net.

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

[10] Jean Tourrilhes, Luiz Magalhaes & Casey Carter. *On-Demand TCP : Transparent peer to peer TCP/IP over IrDA*. Proc. of ICC 2002.

[11] Jean Tourrilhes & Casey Carter. *P-Handoff : A framework for fine grained ad-hoc vertical handoff*. Proc. of PIMRC 2002.

[12] Jean Tourrilhes & Venky Krishnan. *Co-Link configuration : Using wireless diversity for more than just connectivity*. Proc. of WCNC 2003.

[13] Casey Carter, Robin Kravets & Jean Tourrilhes. *Contact Networking: A Localised Mobility System*. Proc. of MobiSys 2003.

[14] Jean Tourrilhes. *BlueTooth roaming proposal*. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Papers/