

Belcarra Technologies

NETWORKING HANDHELDS OVER USB

Benefits and options for implementing and managing a peer-to-peer network of handhelds over existing USB links.

1. Overview

This white paper deals with how to network USB attached devices, handheld devices in particular, to host computers, and some related management issues.

1.1 Executive Summary

The topics are arranged as follows:

- How the communications needs of handhelds have evolved
- Other forms of networking for handhelds
- The need for a distinct form of peer-to-peer networking for handhelds over USB.
- The USBLAN architecture, and its applications, both in improving the customer experience, and developing next generation handheld applications.
- Belcarra's solutions

For this purpose, we may define a *handheld* as a mobile computing platform. It operates in conjunction with an associated desktop (non-mobile) machine which we can term its *desktop* or *base station*. This platform operates in one of three modes

- *synch* Batch operation to exchange (synchronize) data with the handheld
- *detached* The handheld is in the field, unable to communicate with home base (via USB at least)
- *ad hoc* The handheld is connected by a USB link, and an application accesses the link outside the synch application framework.

It is not unreasonable for the handheld to have at least two "home bases" (home and office), and there is no reason to believe the home and office networks are set up similarly. This means that the handheld must adapt to its network environment and avoid use of fixed addresses.

Handheld devices are to be contrasted with *infrastructure devices*, such as ADSL modems, which are also small devices providing access to network infrastructure. The two types of devices differ only in the way they are managed (see for instance 2.5.2 below for details).

1.2 USB as a network port

Recent smart handheld devices offer USB ports for PC/handheld communication. Earlier handheld devices offered serial ports for this purpose, and serial *synchronization* software was created to work over these serial links.

Since USB is a general purpose port, one way of using the USB link on newer devices is make it into a virtual serial port. This allows re-use of existing serial-oriented synch software.

However, it is also possible to use the USB link to turn a collection of USB attached peripherals into a LAN segment, as long as certain network management details are taken care of. To this end Belcarra has created USBLAN architecture, which adapts and improves USB Network standards to allow creation of a LAN segment connecting all handhelds. USBLAN architecture provides for the MAC and IP addresses of a collection of mobile handhelds to be coordinated and distinguished from those of currently connected base station.

Examples are given of how USBLAN architecture frees designers of handheld software (both PC-side and target-side) from the limitations of proprietary API's.

This white paper describes Belcarra's product line, and its implementation of both host and target-side USB Networking solutions for a variety of popular platforms. On the target side, Belcarra's solution are based on the Linux USB Device Framework¹, which is a battle-hardened proven solution for a variety of target architectures. We defer a detailed discussion of products until after the next section, which establishes necessary terminology.

Belcarra offers comprehensive solutions for all USB networked devices, and in particular its solutions that allow a mixture of mobile and infrastructure (cable modem, etc) devices to be attached to the same host computer.

1.3 USB Software Model

Unlike other protocols, such as Ethernet and Serial, USB is a complex layered protocol. USB is conceived as a general purpose computer bus. USB support is divided into layers on both the device (target) and the host (desktop PC).. The USB Software model is shown in Table 1-1, as applied to a networking device. The entities in Column 1 on the device (target) communicate with similarly named entities in Column 3 on the host (base station), using primarily the services of the entity immediately below in the same column, and providing service to the entity immediately above. The entities shown in bold type are provided by Belcarra. Other entities are provided by an operating vendor (OS USB support), or by the device designer (network application).

USB Device		USB Host
Network Application	← Logical Network Connection →	Network Application
OS Network Stack		OS Network Stack
USB Network Function Driver	← Logical Connection →	USB Network Class Driver
OS USB Support		OS USB Support
USB Bus Interface Driver	← Physical Connection →	USB Host Controller Driver

Table 1-1

Since USB is asymmetric, the layers at the host and the device ends have different names. On Microsoft Windows operating systems, USB support is part of the Plug and Play support. Support for common USB host controllers (UHCI and OHCI) is built into Windows, along with class drivers for many simpler devices. Plug and Play find and loads a suitable driver, sometimes asking the user to insert a driver disk.

Correspondingly on the Linux target (device) side, Belcarra engineers have developed the Linux USB Device Framework (which is released under GPL), and some extensions thereto.

Specifically, support for a typical device platform requires the following elements:

- USB Bus Interface Driver compatible with the Device Platform
- USB Network Function Driver that implements the desired protocol
- USB Host Class Drivers for Windows, Linux and Mac OS/X

¹ Some aspects of the Framework are described elsewhere in this white paper, especially in Sections 1.3 and 5.2. For further information, another White Paper on this subject is available on request.

Note that *function driver* (or function module) refers to a target-side driver, whereas *class driver* refers to a host-side driver. A *function driver* is delivered as part of the target firmware. A *class driver* is delivered as a support package to the end user.

The factoring into layers insulates upper layers from peculiarities of layers beneath. So, for instance, the USB Network Function Driver uses the OS USB Support, which is architecture-independent. This reduces the complexity of the network function driver and confines architecture dependent code to the relatively small Bus Interface Driver (see Section 5.6).

In addition to the system components above, host operating system components require a friendly *installer* solution.

1.4 Belcarra USBLAN

With the terminology established in the previous section, we can describe the USBLAN package as follows

- Class drivers for host operating systems:
 - USBLAN Windows Class driver
 - USBLAN for Linux, a Linux class driver (GPL)
 - USBLAN Mac, an OS/X implementation of USBLAN
- Function drivers for targets (peripherals). These function drivers use the GPL Linux USB Device Framework.
 - BLAN network function driver, within the Linux USB Device Framework², all GPL (target network solution suitable for mobile handhelds)
 - CDC/RNDIS network function driver, a proprietary module implementing the CDC and RNDIS protocols (see 3.3.1 below for details on CDC and Section 3.3.2 for information on RNDIS). This module is suitable for infrastructure devices (cable and ADSL modems, etc).

2. Communicating With Handhelds Over USB

USB was conceived as a universal replacement for other types of ports such as serial, parallel, Ethernet, etc. The concept of an interactive device at the end of a USB cable was never considered. Although every USB port is physically identical, each USB Device offers one or more particular *functions* to the host, which *selects* one to be activated. The USB Forum defined various Device Classes to model existing and anticipated devices, such as printers, modems, and Ethernet cards.

In particular, the USB Forum has created a Communications Device Class (CDC), and defined two sub-classes within it of interest, namely CDC Abstract Control Model (CDC ACM), and CDC Ethernet. The first of these models a simple asynchronous serial port (the familiar COM1 to COM4), and the second models a way for a host PC to access an Ethernet segment via USB.

Since a USB device port is only now becoming standard equipment on handhelds, there is legacy software which depends on a serial link. Since the CDC ACM standard provides a (virtual) serial channel, this is a way to re-use serial-oriented management/synchronization software code bases.

However, as handhelds now run general purpose operating systems, they also prefer to run general purpose tools, and have the same facilities, such as wide-area network access, found on other computers.

Therefore, modern handhelds should configure their USB port as a general purpose network (Ethernet) port rather than a serial port, and open up the device to communication with all applications on the PC, not just the special synch software.

² Described more fully in Section 5.2 The Belcarra Linux USB Device Framework

2.1 Expansion of communications needs of handhelds

In the short time that handhelds have been on the market, their capabilities and communications needs have both evolved considerably. At first, synchronization of low-volume personal information over a serial link was adequate. However, three trends are conspiring against serial links.

- Cabling and setting up a serial link is not easy enough for certain classes of users.
- Serial links are not fast or flexible enough for modern applications.
- Serial ports are disappearing from modern desktop computers.

Since serial ports are disappearing, handheld manufacturers are designating USB as the primary way for desktops to communicate with handhelds. To maintain compatibility with older serial-based handhelds and desktops, early management software used USB to emulate a serial link. This was useful for retrofitting USB support to a handheld with no onboard USB port. USB support could be added to a new model of the cradle instead of the unit itself. By this approach, the management solution for the following three cases are the same except for low-level drivers:

- Serial at both ends, no USB
- USB on the host side, USB in the cradle, serial on the handheld
- USB on host and handheld

This approach simply sees the USB port as an exotic serial port as opposed to a general purpose networking port. This view, while convenient to maintain older management software, does little to unlock the true potential of the USB port on a modern handheld.

2.2 Synchronization

As noted in Section 1.2 above, at first all handheld/PC communication was channelled through a special synchronization application. Data gathered on the handheld needed to be backed up on a desktop PC, and conversely data gathered elsewhere needed to be pushed out to the handheld to be available when the handheld was detached from the desktop. So handhelds were seen as normally operating in detached (isolated) mode, and would need to be *synchronized* with the desktop machine whenever convenient (about once a day). At first this information was primarily PIM or *personal information management* data, and its volume, even for the busiest executive, was limited. A standard *synchronization* application was provided to exchange data in a batch manner when the user pressed the *synch* button on the cradle.

Other programs, such as personal finance software (not built in), could, in principle, also synchronize, if they provided an add-on piece of the synch application as well as the application itself on the handheld.

There are two objections to synchronization programs of the past:

- Physical limitations of the serial link
- Funnelling all communications through the synch app's API

2.2.1 Physical limitations of the serial link

Even now, RS232C communications are not simple enough to make a consumer device idiot proof.

- No foolproof way of determining line speed
- No way to tell (at the serial level) if an appropriate device is at the remote end of the link, or if the cable is connected to the appropriate serial port.
- Cabling not completely standard
- High end serial speeds (115kbps) not reliable on all equipment.
- Serial ports non-existent or inaccessible on many computers.

All of these factors make any serial link subject to mysterious failure, and is therefore a customer support issue.

2.2.2 USB only partial solution to RS323 problems

A USB over serial link solves most of these problems.

- The connectors are fully standardized
- There is no serious limit on how many USB peripherals that can be connected
- USB devices are self-identifying, and so management software can be certain that a recognized device is present and active

The USB link is also much faster. However, design limitations in Microsoft Windows, Linux, and other operating systems dictate that the speed of a virtual serial link over USB while faster than traditional serial link speeds does not approach the full capability of the USB link.

- Traditional serial maximum speeds are 115200 (or about 10,000 bytes per second.)
- Serial over USB cannot achieve more than 64,000 bytes per second with standard protocols.

A fully utilized USB link is capable of exceeding 900,000 bytes per second with a full duplex network protocol.

For this reason, this whitepaper considers primarily the network model.

2.2.3 Beyond synch

- Today's handhelds are general purpose platforms. Many of today's networked applications have been adapted to handhelds. Similarly, the handheld may be just one of several small devices available to a user. Therefore, handheld-to-handheld communication may be useful as well.

2.3 Handheld networking media

Modern handhelds are increasingly seen as communications devices, and may have many types of communication channels. For various reasons, Ethernet has never been standard equipment on handhelds, although it is available as an extra-cost add-in card for many platforms. WiFi and Bluetooth look like possible alternatives to USB, but for the foreseeable future, every handheld will need USB Network connectivity in addition to other media it may have.

2.4 Uses for greater speed

As noted earlier, USB can provide a manageable serial solution, but there are issues.

- as noted in Section 2.2.2, this still encumbers the solution with some of the limitations of serial, including poor link speed (typically maximum of 64,000 bytes/second)
- maintaining a serial model forces applications to access the link in a proprietary manner. This adds complexity everywhere, from applications on the handheld to host-based management applications both.

Implementing IEEE 802.3 (Ethernet)³ networking over USB allows for up to 900,000 bytes per second data transfers (depending on the USB Device and USB Host capabilities). allowing totally different paradigms to be used. For instance, this is fast enough to back up the entire contents of a 64MB flash disk in less than 2 minutes. For personal information management, it is fast enough to bring over a new copy of the handheld's PIM database to the host every time there is a synch operation (rather than record-by-record), and then using the full power of the host machine to compare the two. These operations would not be practical at serial speeds.

Ethernet architecture opens the way for applications on the desktop PC to interact more readily and universally with a handheld device. In particular, a Windows CE or Linux-based handheld can implement parts of WINS (Windows Name System) and Common Internet File System (CIFS). For Linux, this can be done using the popular GPL package SMB. In either case, the flash disks of the handheld become visible in the Windows Network Neighbourhood.

From the handheld side, the USB Device can then use all standard networking protocols, applications etc to connect to the desktop, other USB Devices and if necessary through the USB Host to the Internet.

Back in Section 2.2, we referred to two modes handheld operation, *detached* and *synch*. We can now add a third important mode: *ad hoc*. In other words, the network may be used outside the confines of synchronization operations.

³*Ethernet*, as used here, refers to specific aspects of IEEE 802.3 only, namely frame format, management model (decentralized), and all stations equal status and visible to each other. Other aspects are ignored..

2.5 Management of a USB Network

In the Overview for this whitepaper, mention was made of some management issues for USB networks.

Creating a network paradigm opens up great possibilities, and in general allows a wealth of general purpose networking software to be applied to the purpose at hand.

Nevertheless, some attention is needed to network management, as the following sections explain.

2.5.1 Useful aspects of Ethernet for USB networking

Ethernet is as popular for its management model as its physical reality. When the IEEE created a standard framework for various competing aspects of local area networking, it divided the OSI Link (local area network) layer into two sub-layers, named Logical Link Control (LLC) and Media Access Control (MAC). The IEEE 802.2 committee correctly perceived that as long as there was a common addressing structure, details of how frames were transmitted were not extremely important until over-all network performance was considered. Even then, schemes with better performance characteristics (IBM's Token Ring) might lose in the marketplace to simpler schemes (Ethernet).

2.5.2 Two basic types of USB networked device

The CDC Forum's CDC Ethernet models a host PC and a USB device as a tiny 2 station LAN which provides the PC access to a real Ethernet. The host (Windows) machine will normally expect the USB device to provide link management. In other words, Windows will expect DHCP to be running on that link. This is a traditional or *infrastructure* or *fixed mode* device.

Belcarra's USBLAN architecture models the host PC and one or more *handheld* (mobile) USB devices as a multi-station LAN. The PC is the management node here, and need to provide (instead of receiving) management. In other words, USBLAN architecture provides a way for the PC to assign the USB devices compatible Layer 3 (IP) addresses, and for the PC to discover the IP addresses of current attached USB devices.

2.5.3 Address management with USBLAN architecture

USBLAN is a protocol and architecture which both the host and cooperating network devices participate in. Its purpose is to have all the stations on the LAN be visible to each other on the same Layer 3 (IP) LAN segment.

The management objectives, therefore, are:

- Each station should have a unique MAC (Layer 2) address
- IP addresses are on the same IP network block (so that a broadcast will reach all other stations on the segment)⁴
- At least one station already knows or can discover each other's IP address

2.5.3.1 Unique MAC Address

Each USB Device must have a unique MAC address for the network they are being used with. This is a six-byte (48 bit) address that uniquely identifies the device on the network.

Infrastructure devices that are used to connect a desktop system to the Internet must have a globally unique MAC address. This must be composed of an OUI (manufacturers ID) and a three-byte (24 bit) unique address (assigned by the manufacturer.)

Handheld devices are usually part of a small private network (consisting of the desktop PC, its peripherals, and perhaps neighbouring PC's in the same office). Although they should have a globally unique address (same rules as an infrastructure device), a randomly assigned locally administered address has a very low chance of failure. These addresses have the first nibble set to 0x02 and then a 44 bit randomly assigned

⁴ We are generally dealing with private address blocks, so that addresses from differing IP blocks are not routed to each other.

address. The potential for address collision given a 44-bit address among a maximum of 127 devices⁵ is minimal.

2.5.3.2 IP Address Assignment

To be addressable either from the host or from another handheld at the application level, a handheld needs an IP address⁶.

IP addresses must be unique across all the devices that exchange TCP/IP traffic. Creators of synchronization software are accustomed to pre-assigning a handheld peripheral a fixed IP address, such as 192.168.1.100, following IETF RFC 1597⁷, and it requires some effort on both the target and PC side to change this number (if possible at all).

There are three obvious hazards to this approach

- The RFC 1597 address blocks may be in use by the local WiFi access point or cable router
- two devices cannot be managed by the same PC simultaneously
- it is difficult to move the device to a different network with different addressing policies.

Moreover, the host software components performing this function needs to be aware of which peripherals are connected by USB⁸. It follows that this function should be an integral part of USB support rather than performed by a general-purpose application layer DHCP service, such as that available in some editions of recent Microsoft operating systems⁹.

The IP layer uses the Address Resolution Protocol (ARP), to discover which station on a LAN segment is responding to a particular MAC address. This involves a Layer 2 (Ethernet) level broadcast on the segment.

Some infrastructure devices, such as cable routers (which operate downstream from a cable or ADSL modem) would typically require an IP address (a Layer 3 connector). Their function is to relay traffic from one Layer 2 medium (USB) to another (Ethernet), usually with some features such as packet filters.

As noted above, mobile handhelds can no longer be safely assigned fixed IP addresses. However, many other techniques are available for host PC to discover the addresses of its peripherals, and for the peripherals to discover the host and each other.

From the host PC, the USB support layer makes available a list of current attached peripherals.

The standard WINS service can also solve this problem if suitable support is provided in the USB Device (for example using Samba.)

2.6 Using standard software tools to improve time to market.

Applications sell hardware. This is basic.. The handhelds that succeed in the marketplace will have an “edge”, a streamlined method for adapting existing desktop software for the handheld. Trimming desktop software for size is much easier than developing new software. General purpose embedded operating systems, such as Linux and Windows CE are increasing their market share at the expense of VxWorks and other dedicated solutions.

There is an inherent shortage of programming resources to deal with proprietary issues, such as the API's of particular synchronization solutions. Applications on both target and host side require access to standard TCP/IP networking. For instance, assuming the existence of a database solution on the handheld (a suitable

⁵ The maximum number of USB devices that can be connected to a root hub (including hubs).

⁶ Some infrastructure devices, such as cable modems, do not require the USB link to have an IP address at all. Instead, the device is transparent to Layer 3 (IP).

⁷ This RFC reserves certain IP blocks as “unrouteable”, guaranteeing that the public Internet will not put them in the public routing tables. In practise, practically *every* private network, even small household networks, uses these blocks, with a border router performing network address and port (NAPT) translation. This creates a high likelihood of address collision within private networks.

⁸ This allows high level software to know that a certain IP block is for USB attached devices, whereas other RFC 1597 blocks are for other purposes. It also permits all the IP addresses to be assigned in the same block (Class C) rather than separate blocks

⁹ Available in all popular desktop editions of Linux

embedded database), a simple Perl script running under Windows ActivePerl might serve the purpose of a complex proprietary management application written in Visual C++. The enabler of this technique is a general-purpose network link.

Another “bullet point” in modern ease-of-management is *integration into Windows Explorer*. This can be done in two ways. One, the traditional way, is to write a Windows Explorer extension application. A simpler way is to use CIFS/CFS, the cross-platform name for Windows Networking. A Linux-based target can use the GPL package SMB to provide CIFS and make the target “files” browseable, *without deploying any software on the Windows side*. This illustrates the power of using standard networking tools and protocols instead of specialized ones for handhelds.

To the creators of such peripherals, the ease of use and standardized protocols lower initial deployment costs and ongoing support costs by eliminating typical support problems (and costs) that custom high-level software inevitably brings. Even if custom high-level software is necessary for some particular function, the fact that many other functions “just happen” as a result of the compatibility with SMB or other (FTP, NFS, SSH, HTTP, etc.) may reduce the scope and complexity of the proprietary portion.

2.7 Summary

Today’s generation of smart handhelds need sophisticated general purpose networking solutions, built upon existing standards designed for desktop computers. These applications cannot readily use the specialized API’s established for first-generation synchronization and management software. Instead, they demand access to general TCP/IP networking. Since USB is the primary access method for smart handhelds, it follows that the USB support should follow a general purpose peer to peer networking model, i.e. Ethernet. In this way, not only can a handheld and its management PC communicate easily and effectively, but also a PC can administer many handhelds, and handhelds are able to utilize each other.

3. Existing USB Networking Standards and Solutions

Existing USB Networking standards were drawn up with *infrastructure devices* in mind. In other words, the device is an access method to another network. Today’s *mobile* devices are similar, but require management services from the network, as described in Section 2.5.2.

3.1 Examples of infrastructure devices

3.1.1 Ethernet Dongle

In the simplest case, a USB Network adapter is simply an access point to an existing Ethernet segment. It is comparable to a PCI Ethernet card except that there is no need to open the case. The market for these devices is limited because most computers now come with pre-installed Ethernet interfaces, either by a PCI add-in card or on the main board.

3.1.2 Residential broadband device

Some ADSL and Cable modems offer a *USB access method*, meaning that it is possible to use USB instead of Ethernet to access the ADSL or cable network. This access method is especially suited for mobile situations, such as a laptop in a hotel room, when the Ethernet interface, if present, is dedicated to another use (the home or office network)

3.2 Mapping Ethernet concepts onto USB

When people talk about peer-to-peer networking, they generally are referring to Ethernet, the nearly universal form of local area networking, even over media dissimilar to those described in the actual IEEE 802.3 document. In such situations, certain features of a networking protocol are central to identification of a network as “Ethernet-like”.

- Use of IEEE 802.2 Frame and address standards
- Every station has equal status
- Decentralized management model

The first of these requirements is simple enough, although assigning a guaranteed unique MAC address to a handheld is a challenge.

The asymmetry of USB bus can be mostly ignored, allowed the other two requirements to be met as well.

3.3 Existing USB Networking standards

There are two standards in existence for modelling and defining a USB network segment, CDC Ethernet and Microsoft RNDIS

3.3.1 CDC Ethernet

The USB Forum defines an Ethernet profile within the Communications Device Class (CDC). This is known as CDC Ethernet. Its assigned purpose is to model a device, which provides USB access to a remote wired Ethernet segment. This makes it suitable for implementing infrastructure devices.

Neither Microsoft nor Apple supports this standard. They do not supply drivers for this standard on the installation disks of their operation systems, or via updates. The host identifies devices either by a standardized protocol code, or failing this by a vendor/product ID combination. So for instance, there is a standardized USB Audio protocol, and individual USB Audio devices do not require their own driver as long as they enumerate as a standard USB Audio device. A CDC Ethernet driver could support not only a particular CDC Ethernet device, but also all compliant devices without any additional installation of drivers.

In practise, however, vendors of devices using CDC Ethernet must supply a driver to the user. Furthermore, CDC Ethernet is the only networking protocol recognized by the USB Forum. It follows that all other devices (such as RNDIS devices, see below) must have individually installed drivers tied to a vendor/product ID pair (even if all the devices use otherwise identical drivers).

When the target architecture is unable to fully support CDC Ethernet¹⁰, Belcarra has a variant called SAFE.

3.3.2 Microsoft Remote NDIS

Microsoft created Remote NDIS (RNDIS) as a standard way to do networking over USB and similar buses (IEEE 1394 for instance). For the present however, RNDIS is strictly a vendor protocol within the meaning of the USB Forum Communications Device Class Abstract Control Model (ACM), meaning that it sees the USB attached device as a vendor-specific generalized modem, requiring in general a driver installation package for each device (which is recognized by vendor and product ID) even though RNDIS drivers (possibly of early vintage) may be resident already. Microsoft provides materials to create RNDIS kits for all MS operating systems from Windows 98SE forward.

3.3.3 Comparison and evaluation of CDC Ethernet and Microsoft RNDIS

When considering what standard to use to network handhelds, neither CDC Ethernet nor Microsoft RNDIS can be used without modification. As noted, both of these standards are designed with *infrastructure* devices in mind. In particular

- The device is expected to supply the host a MAC (Layer 2) address
- The host handles flow control
- The host needs the device to supply it an IP address (via DHCP)

By contrast, a handheld requires

- A MAC address from the host
- An IP address from the host, either via DHCP or a vendor command
- Manage IP addresses to be distinct but in the same block
- Allow flow control from either end
- Create a 802.1d bridge of multiple devices

Since CDC Ethernet is an open standard, Belcarra selected it as a basis for its handheld-oriented network protocol called BLAN.

¹⁰ The CDC Ethernet standard requires that the UDC¹⁰ support multiple *interfaces*, and at least of the interface must support multiple settings. Several popular handheld processors are unable to support this at the current revision levels.

¹¹ USB Device controller, the USB subsystem of the target CPU

4. USBLAN architecture – Peer to Peer networking over USB

4.1 Desktop Host Requirements

The major three operating systems each have similar requirements:

- The Class driver must be easy to install and operate.
- IP Address assignment to the host interface must support DHCP¹³.
- The Class driver must be able to support both infrastructure and handheld devices.
- The Class driver must differentiate between infrastructure and handheld devices, either using the services offered via the infrastructure device or offering services to handheld devices.
- The Class driver must provide the highest bandwidth and lowest latency possible.

4.2 Handheld Requirements

On the handheld side, a USB solution must implement a network paradigm, and indeed a peer-to-peer network paradigm which allows communication with other peripherals and possibly beyond.

The solution should be easy to implement on all devices. To be part of a peer-to-peer network, the handheld needs to have an address assigned by the host PC. The host PC can pass an IP address to the handheld at enumeration time instead of waiting for the handheld to generate a DHCP request.

4.3 Evolving existing standards to be suitable for handheld devices

CDC Ethernet is an open standard, and can be used as a basis for an improved protocol, which Belcarra calls BLAN.

The requirements for BLAN, beyond those of CDC Ethernet are as follows

- Easy to implement on common target architectures
- Detect whether the target device is mobile or infrastructure. For a mobile device, there should be provision to pass management information to the target. For an infrastructure device, the protocol *accepts* management information from the target (via DHCP).
- Collapse multiple peripherals into a single virtual LAN segment and assign the mobile peripherals a contiguous block of IP addresses on the same Class C.

4.3.1 Coalescing multiple USB links.

This section presents an example of what it is meant by the phrase “assign mobile peripherals a contiguous block of IP addresses”.

For practical purposes, a Class C is the smallest convenient private IP block to manage. Therefore, after 3 peripherals have been attached (using CDC Ethernet or RNDIS) to a Microsoft Windows box, a typical USB driver will create three separate (virtual) Network Adapters. Here is a typical scheme:

Device	MS Windows Address	Device Address
Device 1 (organizer)	192.168.100.1	192.168.100.20
Device 2 (mobile phone)	192.168.101.1	192.168.101.20
Device 3 (specialty handheld)	192.168.102.1	192.168.101.20

We have used 3 private Class C's to create 3 tiny two-node LAN segments. And the peripherals cannot talk to each other. That is the non-USBLAN way. Each USB device has a tiny LAN unto itself.

¹² The DOCSIS (data over cable) standard mandates this.

¹³ Microsoft Windows, for instance, uses DHCP as the default management paradigm of a new network interface. The only alternative is manual management.

A simpler strategy is to use a single Class C, putting the PC at, for instance 192.168.48.20, and all the peripherals starting at 192.168.48.21, and make all Ethernet-level broadcasts visible to all stations.

Device	Address
Device 0 (Windows box)	192.168.48.20
Device 1 (organizer)	192.168.48.21
Device 2 (mobile phone)	192.168.48.22
Device 3 (specialty handheld, such as a mobile messaging terminal)	192.168.48.23

This coalesces all three links to a single 4 node (bridged) fabric. It reduces consumption of private IP blocks and allows peripherals to exchange traffic directly.

4.4 USBLAN architecture

After carefully considering existing standards and the aforementioned management issues particularly concerning handheld/mobile devices, the following characteristics of a useful USB-based networking architecture that meets today's and tomorrow's needs emerge:

- All handhelds (and other cooperating peripherals) should be part of a single virtual multi-drop network segment, with peer-to-peer Ethernet-style networking amongst all stations on the segment, including the desktop system.
- The desktop PC should provide coordinating services to the peripherals, such as IP address assignment.
- The protocol should be readily and efficiently implemented on all major target architectures (Neither RNDIS nor CDC Ethernet meets this requirement).

Belcarra has created an architecture called USBLAN, which meets these requirements. At the USB protocol level, it is an evolution of CDC Ethernet using the Mobile Data Link Model (MDLM). However, it goes beyond the narrower concerns of a single host-to-peripheral link (the concern of all USB Class standards), to create a useful and manageable network solution out of a collection of USB peripherals.

4.5 Belcarra's Comprehensive solution

4.5.1 USBLAN Class drivers for major host OS's

The Belcarra USBLAN product, available for several major host operating systems, implements a USB network (802.3) between the Client and the Host, including all necessary infrastructure to enumerate a new Client, set up its IP address (and assigning it a unique MAC address if the device itself cannot provide one) and set up various facilities and services on the PDA to assist its integration into the host system (time, default route, etc)

All of the Belcarra USBLAN Class drivers allow both infrastructure and handheld type devices to be supported. They are available for:

- Windows
- Linux
- Mac OS/X

4.5.2 USBLAN Function Drivers

Infrastructure devices are supported using either CDC or the Belcarra SAFE protocol. Handheld devices are supported with the Belcarra BLAN protocol.

Belcarra offers three different USB Device Functions:

The network_fd function supports the BLAN protocol

The cdc_fd supports the CDC and SAFE protocols.

The rndis_fd function supports the RNDIS protocol

4.5.3 USBLAN Windows USB Network Class Driver with management facilities

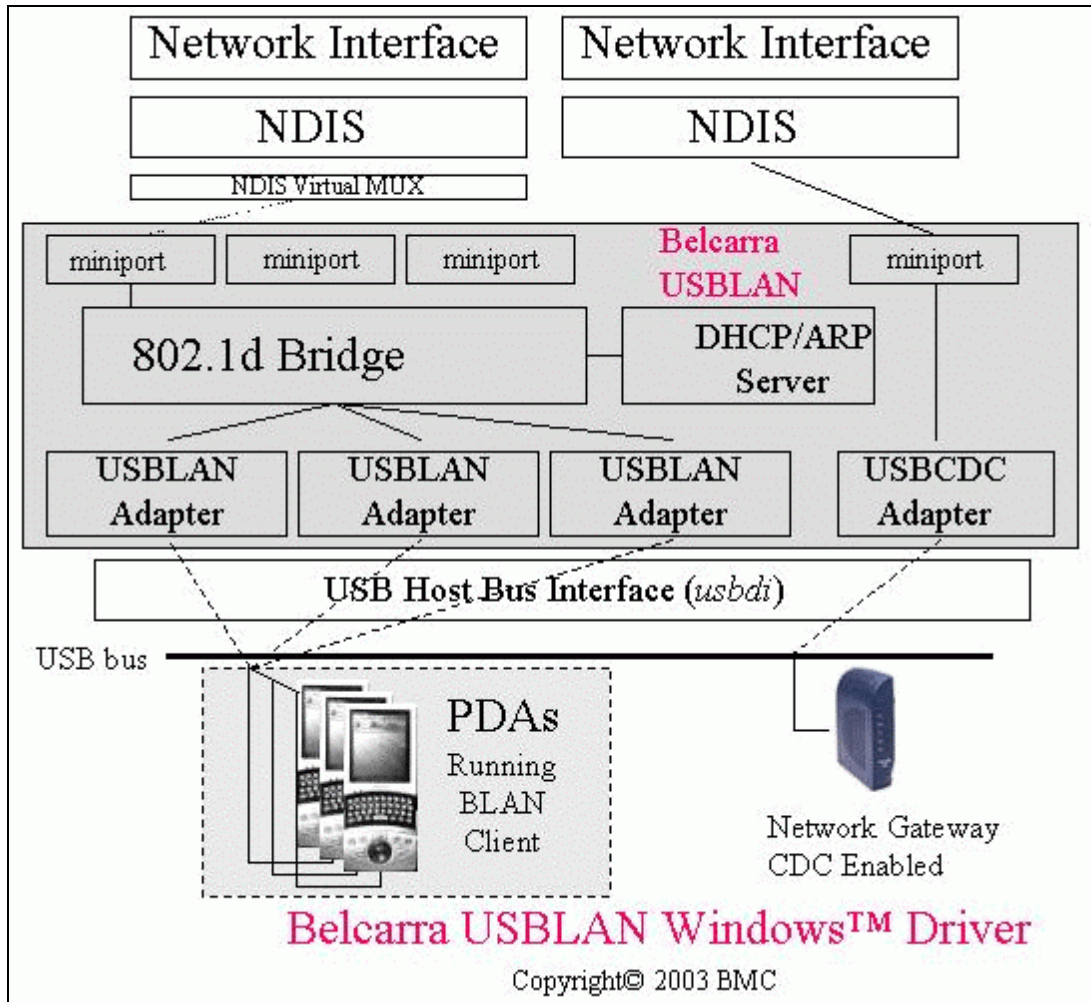


Figure 4-1

In Figure 4-1, the LAN is implemented in the Windows system and enables traffic between the PDA's and the PC. With each separate device being able to send/receive 802.3 type networking data frames to each other device and the host (via the 802.1d bridge in the PC.)

The USBLAN Class drivers automatically detect the type of USB Device that is attached (handheld or infrastructure) and modifies its behaviour appropriately.

- BLAN Device
 - Enrol into bridge
 - Offer IP address and time to device
 - Offer IP address to Windows host (for first or primary interface)
- CDC/SAFE¹⁴ Device
 - Create private network interface
 - Pass DHCP requests from Windows host to USB Device

¹⁴ SAFE is a variant of CDC that can be used for infrastructure devices that cannot meet the technical requirements of CDC.

5. USB Device Architecture

5.1 Proper separation of USB layers on the USB Device

It is important to provide a layered solution on the USB Device.

A USB Device requires two drivers, one driver to implement the Network Function and another driver to interface the USB Device onto the USB via a USB Bus Interface.

By keeping these drivers separated with an appropriate API it is possible to reuse the Function Driver on different architectures and to allow different Function drivers to be interchanged as simply as unloading a driver module and loading another.

Implementation of new drivers and maintenance of existing drivers is also simplified.

5.2 The Belcarra Linux USB Device Framework

The Belcarra Linux USB Device Framework uses the following model:

LAYER	Examples
USB Function Driver	network_fd, cdc_fd or rndis_fd
USB Device Core Layer	usbdcore
USB Bus Interface Driver	au1x00_bi, pxa_bi, dbmx1_bi

This allows for proper separation of the Function portion of the USB Device (networking in this case) and the access to the specific Bus Interface hardware.

Implementing new functions is simplified as it can be tested across a broad range of existing devices.

Implementing new devices is simplified as existing Function and Class drivers are reused and do not need to be modified. This reduces the effort of implementing new devices to a manageable level

5.3 Network Functions For Traditional Infrastructure USB Devices

- cdc_fd - CDC/SAFE – USB standard networking protocol
- rndis_fd – RNDIS – Microsoft proprietary networking protocol

Note that the above two drivers can be implemented as a single device with a dual configuration allowing it to be supported by RNDIS Class drivers under Windows and CDC Class drivers on other OS's.

5.4 Network Functions For Handheld USB Devices

- network_fd – BLAN – Belcarra proprietary networking protocol, part of the USBLAN architecture

5.5 Other Functions For Handheld USB Devices

- acm_fd – ACM – USB Standard serial over USB protocol. Does not require a Windows-side driver.

Some manufacturers, as noted in Section 2.2, may still need to present a virtual serial link for compatibility with existing software. Belcarra supports this requirement with its Abstract Control Model (serial) driver.

5.6 Bus Interface Drivers For Supported USB Devices

- AMD AU1X00 (AU1100, AU1500)
- Hitachi SuperH 7727
- Intel PXA (210, 250, 255)
- Intel StrongArm – SA1100
- Motorola MX1
- SharpSec lh7a400
- Samsung - smdk2500

6. Belcarra Solutions

6.1 About Belcarra Technologies

Belcarra Technologies is the leader in providing high performance network over USB solutions for Linux based devices to connect to Linux, Windows or Macintosh host systems. For more information see www.belcarra.com

6.2 What we offer.

Belcarra offers a comprehensive suite of tools for networking desktop computers running major modern operating systems (MS Windows, Mac OS/X, and Linux) with embedded devices over USB. When the embedded devices run Linux, Belcarra offers both GPL and proprietary solutions within the GPL Linux USB Device Framework, which its engineers created.

On the target side, Belcarra offers support for both infrastructure and handheld devices. There are three basic end-to-end solutions available

Device type	HOST OS	Host USB Support	Target USB support	Protocol
Infrastructure	MS Windows	MS RNDIS	Belcarra rndis_fd	RNDIS
Infrastructure	MS Windows	USBLAN	cdc_fd	CDC Ethernet or SAFE
Infrastructure	Mac OS/X	USBLAN OS/X	cdc_fd	CDC Ethernet or SAFE
Mobile	MS Windows	USBLAN	network_fd	BLAN
Mobile	Mac OS?X	USBLAN OS/X	network_fd	BLAN
Mobile	Linux	USBLAN Linux	network_fd	BLAN

As noted above, the mobile solutions differ in the following ways

- All BLAN nodes are inserted into a IEEE 802.1d bridge
- The host OS assigns a unique IP address (which the target OS is free to ignore)

7. Conclusion

In this paper, we have presented information on how to implement a peer-to-peer network among USB handheld peripherals using Belcarra's USBLAN architecture. We have also presented information on how to use this architecture to simplify the architecture of handheld management software, and indeed to simply use existing parts of Linux and Windows CE in some cases to manage the device.

We also presented Belcarra's comprehensive USB Networking solutions, which support both the novel USBLAN architecture and the conventional CDC architecture on both Linux-based targets, and also Windows, Linux, and Mac-based hosts.

For more information, please contact a Belcarra representative at info@belcarra.com or visit the Belcarra website www.belcarra.com.