2

Speaking NetBIOS

Genuine Imitation

— Well known oxymoron

The hardware part of IBM's PC Network is no longer in use and the protocol that actually ran on the wire is all but forgotten, yet the NetBIOS API remains. Vast sweeping hoards of programs — including DOS itself — were written to use NetBIOS. Like COBOL, it may never die.

Many vendors, eager for a piece of the Microsoft desktop pie, figured out how to implement the NetBIOS API on top of other protocols. There is NetBIOS over DECnet, NetBIOS over NetWare, NetBIOS over mashed potatoes and gravy with creamed corn, NetBIOS over SNA, NetBIOS over TCP/IP, and more. Of these, the most popular, tasty, and important is NetBIOS over TCP/IP, and that's what this chapter is really all about.

NetBIOS over TCP/IP is sometimes called *NetBT* or *NBT*. Folks from IBM — for reasons unfathomable — sometimes call it *TCPBEUI*. NBT is the simplest and most common name, so we'll stick with that.

On the 7-layer OSI reference model, NetBIOS is a session-layer (layer 5) API. Under DOS and its offspring, applications talk to NetBIOS by filling in a record structure known as a Network Control Block (NCB) and signaling an interrupt. The NCBs are used to pass commands and messages between applications and the underlying protocol stack.

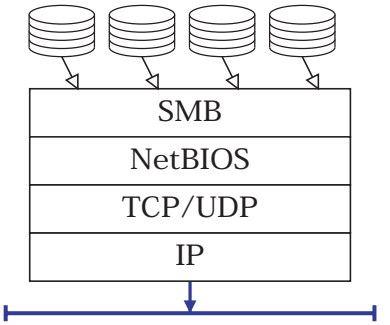

Figure 2.1: *The NetBIOS layer*

The NetBIOS layer is sandwiched between the Server Message Block (SMB) filesharing protocol and the underlying network transport layer.

Fortunately, the NetBIOS API is specific to DOS and its kin. Unix and other systems do not need to implement the NetBIOS API, as there is no legacy of programs that use it. Instead, these systems participate in NBT networks by directly handling the TCP and UDP packets described in two IETF (Internet Engineering Task Force) *Request for Comments* documents: RFC 1001 and RFC 1002 (known collectively as Internet Standard #19). These RFCs describe a set of services that work together to create virtual NetBIOS LANs over IP.

2.1 Emulating "NetBIOS LANs"

At this point, we hit an interesting twist in the terminology. NetBIOS is a driver that presents an API; it is neither a protocol nor a topology. The API does, however, make a number of assumptions about the workings of the underlying network, and it presents some quirky restrictions. The terms "NetBIOS Network" and "NetBIOS LAN" are commonly used to identify the network architecture that is, essentially, *defined* by the NetBIOS API.

RFCs 1001 and 1002 list three basic services that must be supported in order to implement NetBIOS LAN emulation. These are:

- the Name Service,
- the Datagram Service, and
- the Session Service.

The Name Service is used to map NetBIOS names (addresses) to IP addresses in the underlying IP network. The Datagram Service provides for the delivery of NetBIOS datagrams via UDP. Finally, the Session Service is used to establish and maintain point-to-point, connection-oriented NetBIOS sessions over TCP.

2.1.1 *The NetBIOS Name Service*

The NetBIOS LAN architecture is very simple. No routers, no switches — just a bunch of nodes connected to a (virtual) wire. Unlike IP, there is no need for separate hardware addresses, network addresses, or even port numbers. Instead, the communications endpoints are identified by 16-byte strings known as "NetBIOS names."

NetBIOS addressing is dynamic. Applications may add names as needed and remove those names when they are finished. Each node on the LAN will also have a default name, known as the *Machine Name* or the *Workstation Service Name*, which is typically added when NetBIOS starts. The process of adding a name is called *registration*.

There are two kinds of names that can be registered: *unique* and *group*. Group names may be shared by multiple clients, thus providing a mechanism for multicast. In contrast, unique names may only be used by one client per LAN. Keep in mind, though, that these are virtual LANs which may actually be spread out across different subnets in a routed IP internetwork.

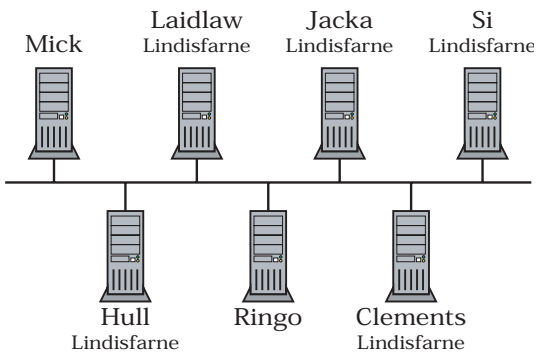

Figure 2.2: *Group names*

In addition to their Machine Names, some of the nodes on this IP LAN have registered the group name `Lindisfarne`. Nodes `Mick` and `Ringo` are not members of the group `Lindisfarne`.

The Name Service is supposed to keep track of all the NetBIOS names in use within the virtual LAN and ensure that messages sent to a given NetBIOS name are directed to the correct underlying IP address. It does this in two ways:

On an IP LAN

Each node keeps a list of the names it has registered (that is, the names it "owns"). When sending a message, the first step is to send an IP broadcast query, called a `NAME QUERY REQUEST`. If there is a machine on the IP LAN that owns the queried name, it will reply by sending a `NAME QUERY RESPONSE`.

So, to send a message to the node which has registered the name `EADFRITH`, the sender calls out "Yo! Eadfrith!" `EADFRITH` responds with an "I am here!" message, giving its IP address.

This is known as "B mode" (broadcast) name resolution, and the participants are referred to as "B **n**odes." In B mode, each node keeps track of — and answers queries for — its own names, so the NetBIOS Name Service "database" is a distributed database.

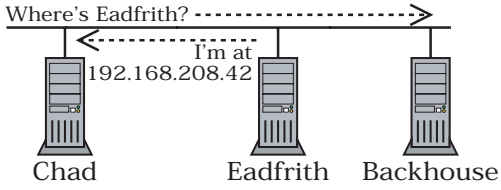

Figure 2.3: *B mode name resolution*

Node `Chad` wishes to contact node `Eadfrith`. Since the underlying transport is IP, `Chad` must first discover the IP address of `Eadfrith`.

`Chad` sends a broadcast name query to all nodes on the local segment, asking `Eadfrith` to respond. All other nodes should ignore the request.

Over a routed Internet

Broadcasts aren't meant to cross subnet boundaries, so a different mechanism is used when the nodes are separated by routers.

The Network Administrator chooses a machine to be the **N**et**B**IOS **N**ame **S**erver (NBNS, aka WINS Server[1]). Typically this will be a Unix host running Samba, or a Windows NT or W2K server. In order to use the NBNS, all of the nodes that are participating in the virtual NetBIOS LAN must be given the server's IP address. This can be done by entering the address in the client's NetBIOS configuration or, on Windows systems, via DHCP.

NBT client nodes send NetBIOS name registrations and queries directly to the NBNS, which maintains a central database of all registered names in the virtual LAN. This is known as "P mode" (point-to-point) name resolution, and its participants are referred to as "P nodes."

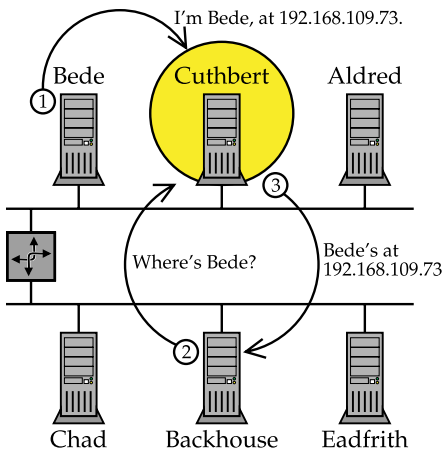

Figure 2.4: *P mode name resolution*

1. Node `Bede` registers its name with `Cuthbert`, the NBNS (WINS server).
2. Node `Backhouse` is looking for `Bede` and sends a query to `Cuthbert`.
3. `Cuthbert` provides the IP address of `Bede` to `Backhouse`.

These are the two basic modes of NetBIOS name resolution over NBT. There are, of course, others. The RFCs describe "M mode" (mixed mode) which combines characteristics of P and B modes. "H mode" (hybrid mode)

1. Microsoft calls their NBNS implementation "**W**indows **I**nternet **N**ame **S**ervice" (WINS). The term WINS is now commonly used instead of NBNS, but we will be pedantic and stick with the latter.

was introduced later; it is similar to M mode except for the order in which B and P mode behavior is applied.

The Name Service runs on UDP port 137. According to the RFCs the use of TCP port 137 can be negotiated for some queries, though few (if any) implementations actually support this.

2.1.2 *The NetBIOS Datagram Service*

In the IP world, TCP provides connection-oriented sessions in which packets are acknowledged, put in order, and retransmitted if lost. This creates the illusion of a continuous, sequential data stream from one end to the other. In contrast, UDP datagrams are simply sent. Thus, UDP requires less overhead, but it is less reliable than TCP. NetBIOS also provides connection-oriented (session) and connectionless (datagram) communications. Naturally, NBT maps NetBIOS sessions to TCP and NetBIOS datagrams to UDP.

The Datagram Distribution Service is the NBT service that handles NetBIOS datagram transport. It runs on UDP port 138, and can handle unicast (also known as "specific"), multicast (group), and broadcast NetBIOS datagrams.

Unicast (specific)

The handling of unicast datagrams is fairly straightforward. The Name Service is used to resolve the destination name to an IP address. The NetBIOS packet is then encapsulated in a UDP packet and sent to the specified IP.

Multicast (group) and broadcast

According to the RFCs, a **B** node can simply encapsulate NetBIOS multicast and broadcast datagrams in UDP and send them to the IP broadcast address. The UDP datagram will then be picked up by all local nodes listening on the Datagram Service port (138/UDP). Thus, NetBIOS broadcast datagrams will reach all nodes in the virtual LAN. In the case of multicast datagrams, nodes which are not members of the group (have not registered the group name) will discard the message.

P, **M**, and **H** nodes are a bit more complicated, as you might expect. When the virtual LAN extends beyond the physical LAN, an IP broadcast will not reach all of the nodes in the NetBIOS name space. In order to deliver group and broadcast datagrams, the NBNS database must be

consulted. How this is (or isn't) actually done will be explained in strikingly painful detail later on. Chapter 5 on page 115 is dedicated to the workings of datagram distribution.

The Datagram Service is probably the second least well understood aspect of NBT, most likely because its correct implementation isn't critical to filesharing. Many implementations get it wrong, and there is much debate over the value of getting it right.

2.1.3 *The NetBIOS Session Service*

The Session Service is the traditional transport for SMB, and this is our primary reason for caring about NetBIOS at all. The Session Service runs on TCP port 139.[2] There is no particular mechanism for multicast or broadcast because each session is, by definition, a one-to-one connection. The RFCs do, however, briefly discuss what might happen if a session setup request were sent to a group name (see RFC 1001, Section 16.1.1.2).

We will get to the details of session creation, use, and closure when we discuss Session Service implementation.

Weirdness Alert

TCP/138 has no defined behavior under NBT and Microsoft never implemented support for NBT Name Resolution over TCP/137, yet some versions of Windows seem to listen on these two TCP ports when NBT is active.

```
C:\> netstat -a

Active Connections

  Proto  Local Address      Foreign Address  State
  TCP    paris:137          PARIS:0          LISTENING
  TCP    paris:138          PARIS:0          LISTENING
  TCP    paris:nbsession    PARIS:0          LISTENING
  UDP    paris:nbname       *:*
  UDP    paris:nbdatagram   *:*
```

It turns out that this is due to a known bug in the `netstat` *utility included with older Windows releases.*

2. If the NBT authors had used TCP/138 instead of 139 for the Session Service, they could have saved a couple of ports. Instead, TCP/138 and UDP/139 are wasted.

Email

```
From: Jean-Baptiste Marchand
  To: Chris Hertel

Hello,

I've noticed that, in Section 2.1, there is a weirdness alert about
the Windows netstat program showing TCP ports 137 and 138 open,
whereas only UDP ports 137 and 138 are actually opened by the NetBT
driver.

In fact, it is a known problem in Windows NT (this is fixed in
Windows 2000 and later) that netstat shows TCP ports opened, whereas
only UDP ports with the same number are opened. This is documented
in an entry of Microsoft's knowledge base (Q194171).

This article states that this is only a display problem. This is
true and can be verified using any TCP port scanner.
```

2.2 Scope: The Final Frontier

This is a good point at which to get up, stretch, make a nice hot cup of tea for yourself, take a soothing bath, play with your cat, go for a long walk in the park, take dance lessons, volunteer in your community, sort and organize your old photographs, or join a United Nations Peace Keeping Force. The Datagram Service was previously described as "the second least well understood aspect of NBT." Guess which bit wins first prize.

Scope is an oddity of NBT, not because it was a bad idea (though perhaps it was) but because few have ever bothered to really understand it. In practice this feature is rarely used, in part because it is rarely implemented to its full potential.

In the RFCs, the term *scope* is used as a name for:

- the set of NetBIOS nodes that participate in an NBT virtual LAN,
- an identifier used to distinguish one virtual LAN from another, and
- that which is included within the purpose of the RFC document.

...but the last of these is beyond the scope of this discussion, so let's take a closer look at the first two.

Scope is explained in RFC 1001, Section 9, which starts off by saying:

```
A "NetBIOS Scope" is the population of computers across
which a registered NetBIOS name is known. NetBIOS broadcast
and multicast datagram operations must reach the entire extent
of the NetBIOS scope.
```

This basically means *all nodes connected to the virtual LAN*. So, for B nodes the NetBIOS scope consists of all nodes within the local IP broadcast domain that are running NBT. For P nodes, the NetBIOS scope includes all nodes across the routed internetwork that run NBT and share the same NBNS. For an M or H node, the scope is the union of the local broadcast and the NBNS scopes.

This is all quite straightforward when all NBT nodes are of the same node type, but strange things can happen when you mix modes, particularly in a routed environment.

P & B

Two separate scopes are defined. The B nodes will only see other B nodes on the same wire, and the P nodes will only see other P nodes using the same NBNS. If creating separate NetBIOS vLANs is your goal, then mixing P and B nodes on the same wire is perfectly okay.

P & M

This results in a single scope. The M nodes perform all of the functions of a P node, including registering their names with the NBNS. Thus, all P nodes can see all M nodes, though M nodes on the same wire can bypass the NBNS when resolving names.

B & M

On a single, non-routed IP LAN there will be only one scope. The M nodes will register and resolve names via the broadcast mechanism, making their use of the NBNS pointless.

Things start going terribly wrong, though, when the NetBIOS vLAN is distributed across multiple subnets in a routed internetwork. When this happens the result is multiple, intersecting scopes. B nodes on one subnet will not be able to see any node on any other subnet. M nodes will see all other M nodes, but only the B nodes on their local wire. Thus,

parts of the NetBIOS vLAN are hidden from other parts, yet all are somewhat connected via the common M node scope.

One result of this mess is the potential for name collisions. A B node could register a name that is already in the NBNS database, and an M node might register a name that one or more B nodes on remote subnets have already claimed. Name resolution then essentially fails because the same name does not resolve to the same IP address across the fractured scope.

The RFCs recognize this potential for disaster and warn against it. See RFC 1001, Section 10.

P, B, & M

From bad to worse. The P nodes can see all of the M nodes which can see some of the B nodes which cannot see any P nodes at all. B nodes and M (or H) nodes don't mix.

We now have a good handle on our first definition of scope: "*the set of NetBIOS nodes that participate in a virtual LAN*." What about the second: "*an identifier used to distinguish one virtual LAN from another*"? (This is a good point at which to get up, stretch, make a nice hot cup of tea for yourself...)

Every scope has a name, called the Scope Identifier (Scope ID). The most common Scope ID is the empty string: `""`. Indeed, this is the default in Windows, Samba, jCIFS, and every other system encountered so far. The only problem with this name is that it becomes too easy to forget that the Scope ID exists.

We have already seen that distinct NetBIOS vLANs can be created by using the behavior of B, P, M, and H nodes to create separate scopes. For example, multiple scopes are defined when multiple independent NBNS's provide service for P nodes. B nodes on separate IP LANs are also in separate scopes, and so on. The Scope ID provides another, more refined mechanism for separating scopes.

Think of an IP LAN with a bunch of B nodes. Some of the B nodes have Scope ID `DOG`, and others have Scope ID `CAT`. Only members of scope `DOG` will listen to messages sent with that ID; the cats will ignore messages sent to the dogs. So, even though all of the B nodes are on the same wire, we have two separate scopes. The same applies to P and M nodes. The Scope IDs identify, and separate, virtual NetBIOS LANs. Note, though, that an NBNS will handle

requests from any node regardless of scope. A single NBNS server can, therefore, support multiple scopes.

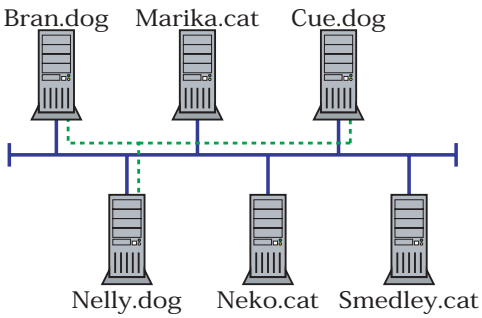

Figure 2.5: *Multiple named scopes in a broadcast LAN*

Nodes with scope ID DOG are in their own virtual NetBIOS LAN. Nodes with scope ID CAT will ignore broadcasts to the DOG scope.

According to RFC 1001/1002, a node may belong to more than one scope. In practice, however, it is much easier to choose a single scope and stick with it. This is particularly true for DOS and Windows systems because NetBIOS itself has no concept of scope. The Scope ID is a feature of NBT, and programs that call the NetBIOS API have no way of telling NBT which scope to use.

The RFCs suggest that extensions might be added to NetBIOS to manage scope, but using those extensions would require changes to applications. Further, other NetBIOS transports would not support extensions, which would result in compatibility problems.

Confusion Alert

Scope IDs are used by the Name Service and the Datagram Service, but not *the Session Service. This seems awkward at first, but it makes sense when you consider that the NetBIOS API itself has no knowledge of scope.*

Once again, Scope IDs serve only to identify virtual NetBIOS LANs. They operate at a lower level than the NetBIOS API.

2.3 Thus Endeth the Overview

Now that you have a clear and precise understanding of the workings of NetBIOS over TCP, go read RFC 1001. That ought to muddy the waters a bit. Clear or not, the next step is to write some code and see what works — and what doesn't. Actual implementation will provide a lot of opportunity to discuss details, bugs, and common errors.