5

The Datagram Service in Detail

If a tree falls in the forest,
and no one is there to hear it...

Let's drill home this key concept one more time:

> NBT provides a set of services which combine to create a virtual NetBIOS LAN over TCP/UPD/IP transport.

This would be a senseless thing to do *except* for the fact that lots of software uses (or used to use) the NetBIOS API. The whole point is to maintain the form and function of the API while completely replacing the guts and machinery which lie beneath. This point gets lost, however, when we deal with systems that are not derived from MS-DOS and have no use for NetBIOS itself. On these systems we work directly with the guts of NBT and, therefore, are easily confused by the odd behavior of the machinery.

So, to provide a little context, here are the four NetBIOS API functions which the Datagram Service was designed to support:

- `Send Specific Datagram`
- `Receive Specific Datagram`
- `Send Broadcast Datagram`
- `Receive Broadcast Datagram`

Let's start by looking at the two `Send Datagram` functions. These two API calls provide us with three transmission options: unicast, multicast, and broadcast. Here's how they work:

Send Specific Datagram

This function requires a NetBIOS name as a parameter.

- If the name is unique, the datagram is *unicast*.
- If the name is a group name, then the datagram is *multicast*.

Send Broadcast Datagram

This function does not accept a NetBIOS name. Broadcast datagrams are sent the length and breadth of the NetBIOS LAN, and picked up by any node that is listening.

That was easy. Now let's look at what happens when we map those functions onto UDP/IP at the NBT layer...

Send Specific Datagram

A `NAME QUERY REQUEST` is issued to discover whether the destination name is a unique or group name.

- If it is a unique name, then the message can be encapsulated in a UDP packet and sent to the IP address given in the `NAME QUERY RESPONSE`. In NBT terminology, this is a `DIRECT UNIQUE DATAGRAM`.
- If it is a group name...
  - If the sender is operating in B mode, it will broadcast the packet on the local IP subnet so that all group members can receive it.
  - If the sender is *not* operating in B mode, then the datagram is forwarded to the **N**et**B**IOS **D**atagram **D**istribution Server (NBDD).

  In NBT terminology, a multicast datagram is known as a `DIRECT GROUP DATAGRAM`.

Send Broadcast Datagram

The wildcard name (with the sender's Scope ID appended) is used as the destination name.

- If the sender is operating in B mode, it will broadcast the packet on the local IP subnet. All NBT nodes within the same scope will be able to receive the message.
- If the sender is *not* operating in B mode, then the datagram is forwarded to the NBDD.

As you can see from the description, unicast datagrams are easy, B mode is easy, but handling multicast or broadcast in P, or M, or H mode is a bit more complicated. We'll give that topic a section heading all its own, just to show that it is a fairly hefty chunk of tofu.

5.1 Datagram Distribution over Routed IP Internetworks

The **N**et**B**IOS **D**atagram **D**istribution Server (NBDD) compliments the NBNS. It assists in extending the virtual NetBIOS LAN across a routed IP internetwork by relaying multicast and broadcast NetBIOS datagrams to nodes on remote subnets. The NBDD's job is to make sure that the datagrams get to where they're supposed to go. It works something like a lawn sprinkler — one input leads to a spray of outputs. Here's what happens:

- A P (or M or H) node sends a datagram to the NBDD.
- The NBDD consults the NBNS database and gathers the IP addresses of all intended recipients.
- The NBDD then sends a copy of the message, via unicast UDP datagrams, to each of the IP addresses in the list.

That seems simple enough, but we claimed earlier that the Datagram Service is the second least well understood aspect of NBT. What are we missing?

A closer inspection reveals an obvious problem. If the number of destination nodes is large, a whole bigbunch of traffic will be generated — possibly enough to bring the NBT virtual LAN to its knees (which might really, really annoy people). The NBDD design will work well enough for small, trusted networks but it simply does not scale.

Another problem is that RFC 1001 offers a loophole for implementors: the NBDD is permitted to silently ignore requests to relay datagrams. If, for any reason (including laziness on the implementor's part) the NBDD will not

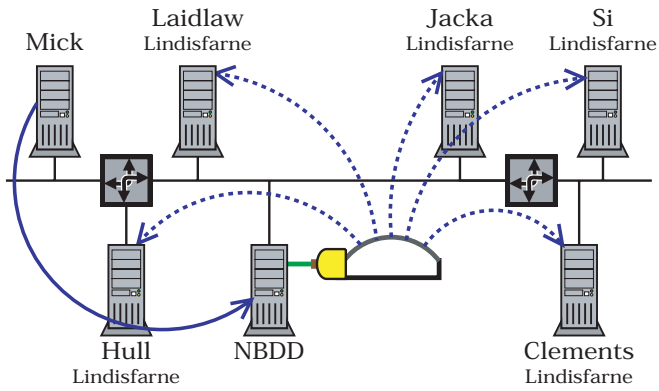


Figure 5.1: *The Datagram Distribution Server*

Node `MICK` wants to send a message to all members of group `LINDISFARNE`, but the NetBIOS LAN is distributed across a routed IP internetwork. `MICK` sends the datagram to the NBDD which relays the message to all group members.

or can not relay a datagram, it simply discards the message without sending any sort of error message back to the originating node.

This loophole might make the NBDD so unreliable as to be useless, except that the Datagram Service also supports a query operation that allows the client to ask the NBDD whether or not it will relay a message. If the NBDD answers the query with a "yes", then the client can send the datagram with the assurance that it will be relayed to all intended recipients. A negative reply means that the NBDD will not relay the message.

Reminder Alert

Datagrams are not considered reliable. As with the UDP service in an IP network, it is expected that some NetBIOS datagrams may be lost.

By allowing the NBDD to silently discard datagrams, however, the lack of reliability is amplified well beyond what would be expected from simple network packet loss.

One more monkey-wrench to throw into the works... Given a multicast (not broadcast) datagram, if the NBDD will not relay the message, the client can give it another shot. The client has already performed a Name Service `NAME QUERY REQUEST` operation, and received a `NAME QUERY RESPONSE` from the NBNS. It did this to determine that the destination name was, in fact, a group name rather than a unique name. If the NBNS is RFC-compliant, then the `NAME QUERY RESPONSE` will contain a list of all

the IP addresses of the group members. If the NBDD won't relay the message, then the client can unicast the datagram to each entry in the list.

To summarize:

- Unicast datagrams are simply sent to the intended recipient.
- In B mode, multicast/broadcast datagrams are broadcast on the local LAN.
- For multicast/broadcast datagrams in P, H, and M modes the NBDD should be queried to see if will relay the datagram.
  - If a positive response is received, then send the datagram to the NBDD for distribution.
  - Else:
    - If the datagram is multicast and the Name Server returned a complete IP list, then send the message via unicast datagrams to each IP in the list.
    - Else, broadcast the datagram on the local subnet and hope that some nodes will receive it.

Confused? Don't be surprised if you are. It isn't a pretty system... and it gets worse. Because of the potential network problems and the awkwardness of the design, very few implementations even try to match the RFC specification. Unfortunately, no one has come up with a better solution... which means that what actually exists in the wild is worse than what was just described.

5.2 The NBDD and the Damage Done

To be blunt, Microsoft messed up the Datagram Service. Their NBNS implementation (WINS) does not report all of the IP addresses associated with a group name. Instead, group names are mapped to a single IP address — the limited broadcast address: 255.255.255.255. This is contrary to the RFCs, and it causes a few problems.

Without a list of IPs for each group name, the NBDD cannot be implemented at all. With no NBDD and no IP list, there is no way to ensure that multicast and broadcast datagrams will be sent to all group members. This breaks the continuity of the NetBIOS virtual LAN. On a "real" NetBIOS LAN, a message sent to a group name would actually reach all members of that

group. That is what should happen under NBT as well, but it doesn't. The best that can be done is to broadcast on the local subnet, in which case *some* members of the group *may* get the message.

Microsoft must have realized their mistake, because they later created what they call "Internet Group" names (also called "Special Group" names). For names in this category, WINS comes close to behaving like a proper NBNS; it will store up to 25 IP addresses per name, deleting the oldest entry to make room if necessary. For these names, a `POSITIVE NAME QUERY RESPONSE` from a WINS server will list up to 25 valid IP addresses.

Internet Group names are identified by their suffix. Originally only group names with the `0x1C` suffix were given special treatment, but more recently (with W2K?) group names with a suffix value of `0x20` can be defined as having Internet Group status. Note that unique names may also have these suffixes but, since they are not group names, no special handling is required.

Sadly, most non-Microsoft implementations (including Samba) follow Microsoft's example. They map group names to the 255.255.255.255 IP address, store only 25 IPs for Special Group names, and fail to implement the NBDD.[1] This can cause trouble for some clients (OS/2, for example) which expect RFC behavior.

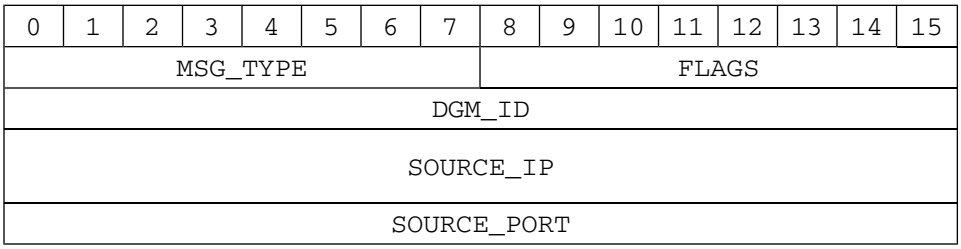
Sigh.

5.3 Implementing a Workable Datagram Service

That last section was a bit of a rant. Sorry. Time to pick up the pieces and move on. Let's talk about how the parts that work actually work.

As with the Name Service, each Datagram Service packet has a header. The datagram header is 10 bytes long, arranged as follows:

1. Network Telesystems, which has since been acquired by Efficient Networks, used to have an NBNS implementation that handled group names correctly and worked quite well with IBM's OS/2. Brian Landy has also written a set of patches for Samba's nmbd daemon which provide more complete NBDD support. See `http://www.landy.cx/`.



Here is a quick rundown of the fields:

MSG_TYPE (1 byte)

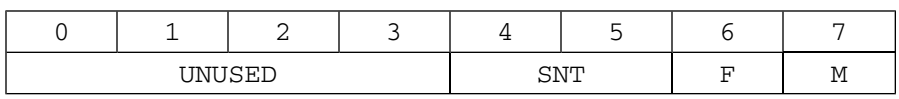
This field is something like the OPCODE field in the Name Service header. It indicates which type of Datagram Service message is being sent. It has the following possible values:

```
0x10 == DIRECT_UNIQUE DATAGRAM
0x11 == DIRECT_GROUP DATAGRAM
0x12 == BROADCAST DATAGRAM
0x13 == DATAGRAM ERROR
0x14 == DATAGRAM QUERY REQUEST
0x15 == DATAGRAM POSITIVE QUERY RESPONSE
0x16 == DATAGRAM NEGATIVE QUERY RESPONSE
```

The first three of these represent unicast, multicast, and broadcast datagrams, respectively. The DATAGRAM ERROR packet is used to report errors within the Datagram Service. (There are only three errors defined in the RFCs.) The final three types are used in the query mechanism described earlier.

FLAGS (1 byte)

This field breaks down into a set of bitwise subfields:



SNT: Sending *Node Type*

This subfield has the following set of possible values (in binary):

00 == B node
01 == P node
10 == M node
11 == NBDD

Microsoft did not implement the NBDD. They did, however, introduce H mode. In practice the value 11 is used to indicate that the sending node is an H node.

F: FIRST flag

Indicates that this is the first (and possibly only) packet in a fragmented series.

M: MORE flag

Indicates that the message is fragmented, and that the next fragment should follow.

The F and M flags are used to manage fragmented messages, which will be described in more detail real soon now.

DGM_ID (2 bytes)

The Datagram ID is similar in purpose to the NAME_TRN_ID field in Name Service headers. There should be a unique DGM_ID for each (conceptual) call to the NetBIOS Send Specific Datagram or Send Broadcast Datagram functions. More about this when we discuss fragmented messages.

SOURCE_IP (4 bytes)

The IP address of the *originating* node. If the datagram is being relayed via the NBDD, then the IP header (at the IP layer of the stack, rather than the NBT layer) will contain the IP address of the NBDD. The SOURCE_IP field keeps track of the IP address of the node that actually composed the datagram.

SOURCE_PORT (2 bytes)

As with the SOURCE_IP field, this field indicates the UDP port used by the originating node.

The above fields are common to all Datagram Service messages. RFC 1002 includes two more fields in its header layout: the DGM_LENGTH and

PACKET_OFFSET fields. It is kind of silly to have those fields in the header, as they are specific to the messages which actually carry a data payload: the DIRECT_UNIQUE, DIRECT_GROUP, and BROADCAST DATAGRAM message types.

Since the DGM_LENGTH and PACKET_OFFSET fields are associated with the datagram transport messages, we will break with tradition and put those fields together with the message structure. Here is a record layout:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DGM_LENGTH															
PACKET_OFFSET															
SOURCE_NAME															
DESTINATION_NAME															
USER_DATA															

DGM_LENGTH (2 bytes)

The formula given for calculating the value of the DGM_LENGTH field is:

```
DGM_LENGTH = length( SOURCE_NAME )
           + length( DESTINATION_NAME )
           + length( USER_DATA )
```

That is, the number of bytes following the PACKET_OFFSET field.[2]

PACKET_OFFSET (2 bytes)

Used in conjunction with the F and M flags in the header to allow reconstruction of fragmented NetBIOS datagrams.

SOURCE_NAME (34..255 bytes)

The encoded NBT name of the sending application. Recall that NBT names are communication endpoints in much the same way that a UDP or TCP port is an endpoint. The correct SOURCE_NAME must be supplied to identify the service or application that sent the datagram.

2. This field is probably not even used by most implementations. For a long time, Samba miscalculated the DGM_LENGTH field by including the length of the 14-byte RFC header. This bug (fixed as of 2.2.4) did not seem to cause any trouble.

DESTINATION_NAME (34..255 bytes)

The encoded NBT name of the destination application or service. For broadcast datagrams, the DESTINATION_NAME will be the wildcard name — an asterisk ('*') followed by fifteen nul bytes. The NBT name must include the Scope ID (even if it is the default empty scope, "").

USER_DATA (0..512 bytes)

The actual data to be transmitted.

That's basically all there is to it, with the exception of fragmentation. The example packet below describes an unfragmented message.

```
DATAGRAM_HEADER (unfragmented)
  {
  MSG_TYPE = <10 = unicast, 11 = multicast, 12 = broadcast>
  FLAGS
    {
    SNT = <Node type, as shown above>
    F   = TRUE   (This is the first in the series)
    M   = FALSE  (No additional fragments follow)
    }
  DGM_ID      = <Datagram identifier>
  SOURCE_IP   = <IP address of the originating node>
  SOURCE_PORT = <Originating UDP port>
  }
DATAGRAM_DATA
  {
  DGM_LENGTH       = <Name lengths plus USER_DATA length>
  PACKET_OFFSET    = 0
  SOURCE_NAME      = <Fully encoded NBT name of the sender>
  DESTINATION_NAME = <Fully encoded NBT name of the receiver>
  USER_DATA        = <Datagram payload>
  }
```

Some quick notes:

- The DGM_ID should be unique with respect to other active datagrams originating from the same source. With 64K values from which to choose, this should not be difficult.
- Once again, the SOURCE_IP, SOURCE_PORT, and SOURCE_NAME are all relative to the originator of the datagram. An NBDD does not alter these fields when it relays a message.

- NBT datagrams are sent *within* scope. The Scope ID must be present in the SOURCE_NAME and DESTINATION_NAME fields, even if it is the empty scope ("").

5.3.1 *Fragmenting Datagrams*

A little way back, we mentioned the NetBIOS API `Send Specific Datagram` and `Send Broadcast Datagram` functions. These functions each accept up to 512 bytes of data on input. Given that number, the maximum on-the-wire size of an NBT datagram is:

```
   10 bytes (Header)
+   2 bytes (DGM_LENGTH)
+   2 bytes (PACKET_OFFSET)
+ 255 bytes (maximum size of SOURCE_NAME)
+ 255 bytes (maximum size of DESTINATION_NAME)
+ 512 bytes (maximum size of USER_DATA)
-----
 1036 bytes
```

and that, of course, does not include the UDP and IP headers. The whole thing is fairly chunky — easily more than double the size of the data actually being sent.

The RFC authors were concerned that the UDP transport might not carry datagrams that big, so they provided a mechanism for breaking the USER_DATA into smaller fragments, like so:

first fragment

```
FLAGS.F       = TRUE (This is the first fragment)
FLAGS.M       = TRUE (Additional fragments follow)
PACKET_OFFSET = 0
```

middle fragments

```
FLAGS.F       = FALSE (This is the not the first fragment)
FLAGS.M       = TRUE  (Additional fragments follow)
PACKET_OFFSET = <Data offset of fragment>
```

final fragment

```
FLAGS.F       = FALSE (This is not the first fragment)
FLAGS.M       = FALSE (No more fragments follow)
PACKET_OFFSET = <Data offset of fragment>
```

The value of the PACKET_OFFSET field is the sum of the lengths of all previous fragments. This value is included in the message so that the receiver can keep the fragments in sync as it rebuilds the original USER_DATA. This is necessary, because datagrams do not always arrive in the order in which they were sent.

Now that you have learned all of that, you can forget most of it. As is typical for the Datagram Service, the fragmentation feature is rarely — if ever — used. The IP layer has its own mechanism for handling large packets so NBT datagram fragmentation is redundant.

It is possible that someone, somewhere, has implemented fragmentation, so an NBT implementation should be prepared to deal with it. One simple option is to discard fragments. This can be considered valid because the Datagram Service is considered "unreliable."

Something else to keep in mind: The 512-byte maximum size for the USER_DATA field is required at the NetBIOS layer, but not the NBT layer. Since the NetBIOS API is not required for implementing NBT, you mustn't expect that the datagrams you receive will fit within the limit. Code defensively.

5.3.2 *Receiving Datagrams*

NBT receives datagram messages on UDP port 138, so clients must listen on that port at the UDP level. When a message datagram is received (MSG_TYPE is one of 0x10, 0x11, or 0x12) the DESTINATION_NAME is checked against the local name table. If the name is not found, the client should reply with a DATAGRAM ERROR MESSAGE. The available error codes are:

```
0x82 == DESTINATION NAME NOT PRESENT
0x83 == INVALID SOURCE NAME FORMAT
0x84 == INVALID DESTINATION NAME FORMAT
```

The first value is used whenever the DESTINATION_NAME is not in the local name table at the receiving end. The other two codes are sent whenever the source or destination NBT names, respectively, cannot be parsed.

If the name is found in the local table, then the datagram may be passed to any application or service that is listening for the given DESTINATION_NAME. The NetBIOS API provides the Receive Specific Datagram and Receive Broadcast Datagram calls for this purpose.

If there are no Receive Datagram requests waiting, the datagram is quietly discarded.

NBDD processing (for those bold enough to want to implement an NBDD) is similar. When the NBDD receives a datagram it will search the NBNS database instead of the local name table. Error messages are returned as above for missing or malformed names.

One more note: As a safety precaution, the receiving node should probably verify that the SOURCE_IP field in the datagram header matches either the source address in the IP header, or the NBDD address (if there is one).

5.3.3 *Querying the NBDD*

The NBDD query message is simply an NBT Datagram Service header with the DESTINATION_NAME appended:

```
DATAGRAM_HEADER
  {
  MSG_TYPE = 0x14 (DATAGRAM QUERY REQUEST)
  FLAGS
    {
    SNT = <Node type>
    F   = TRUE
    M   = FALSE
    }
  DGM_ID      = <Datagram identifier>
  SOURCE_IP   = <IP address of the originating node>
  SOURCE_PORT = <Originating UDP port>
  }
DATAGRAM_DATA
  {
  DESTINATION_NAME = <Encoded NBT name of the intended receiver>
  }
```

If there is an NBDD, and if it can relay the request, it will change the MSG_TYPE field to 0x15 (POSITIVE QUERY RESPONSE) and echo the packet back to the sender. If the NBDD is unwilling or unable to relay the message it will set MSG_TYPE to 0x16 (NEGATIVE QUERY RESPONSE) before sending the reply.

5.3.4 *The Second Least Well Understood Aspect of NBT*

It really should have been much simpler, but given the design flaws and implementation errors it is no wonder people have trouble with the Datagram Service. Our hope is that this section has cleared things up a bit, and explained the problems well enough to make them easier to solve.

Just to finish up, here are a few tips:

- The NBDD should never relay datagrams to itself. If the NBDD host is also an NBT end node, then it must deliver datagrams to itself *and then pass them along to the NBDD*. There is no way to know if a received datagram is intended for the end node or the NBDD.
- Likewise, if a host is acting as both end node and NBDD, the end node processing should *not* generate `DESTINATION NAME NOT PRESENT` (`0x82`) errors. The datagram should be passed along to the NBDD instead.
- The NBNS should store all IP addresses associated with a group name. If necessary, it can return the local broadcast IP address (255.255.255.255) in response to name queries, thus maintaining compatibly with Microsoft's WINS. Storing all group name IP addresses is necessary for NBDD implementation.
- Set a limit on the size of the IP list to which an NBDD will relay messages.
- Don't worry about it. If you get the basics right, your system will work well enough. Very few systems expect a complete and proper NBT Datagram Service implementation.