

# NBD NOTES

Week 1 28/06/2023

Thodoris Pontzouktzidis

Network Block Device: Forwarding block device in linux client-server (Standard protocol) TCP  
client <-> network <-> server

## Please keep in mind:

### Reading through:

- 1) <https://github.com/NetworkBlockDevice/nbd/blob/master/doc/proto.md>  
This directory contains developer documentation. It's probably not useful unless you wish to help with implementing nbd.

This file tries to document the NBD protocol as it is currently implemented in the Linux kernel and in the reference implementation. The purpose of this file is to allow people to understand the protocol without having to read the code. However, the description above does not come with any form of warranty; while every effort has been taken to avoid them, mistakes are possible.

```
1   Here's a (probably incomplete) list of things I think need to be done to
2   nbd. If you feel like taking one of these up, I owe you beer if we ever
3   meet...
4
5   - Add (optional) authentication to the protocol. Probably best to use
6     SASL to implement this. I should like to have Kerberos working, too,
7     though that might be somewhat involved to get right.
8     Not sure whether we would need full per-export authentication, or if
9     an on/off switch in an export (and the full configuration in the
10    generic section) would be enough
11  - Have support for setting defaults for exports in the generic section.
12  - Turn much of nbd-server into a library, with the server itself just
13    being a stub that reads the config file and exports files.
14  - Performance improvements: nbd-server should use sendfile() and/or
15    libevent to make things go faster. This should be extensively tested
16    so we're sure things are actually going faster.
17  - ... probably more, but I can't remember much of them right now. I'll
18    add to this list as I remember things.
```

- 2) <https://github.com/NetworkBlockDevice/nbd> README FILE

**WILL HELP US SETUP AND RUN WITH NBD.**

- 3) <https://www.youtube.com/watch?v=PMa6KFX9AxM> This video can also help understand the protocol.

# Start of proto.md file

## Protocol phases.

Client kernel (or userspace) driver forwards request to server where it is processed by a userspace program.

2 phases handshake , transmission

## Handshake.

A connection is established and an exported NBD device along other protocol parameters are negotiated between the client and the server.(1)

After (1) client performs in **userspace**:

```
ioctl(nbd, NBD_SET_SOCKET, sock)
ioctl(nbd, NBD_DO_IT)
```

From Handshake phase to transmission phase

**nbd = file descriptor for an open /dev/nbd0 device node**

When handling the client-side transmission phase with the Linux kernel, the socket between the client and server can use either Unix or TCP sockets. For other implementations, the client and server can use any agreeable communication channel (a socket is typical, but it is also possible to implement the NBD protocol over a pair of uni-directional pipes). If TCP sockets are used, both the client and server SHOULD disable Nagle's algorithm (that is, use `setsockopt` to set the `TCP_NODELAY` option to non-zero), to eliminate artificial delays caused by waiting for an ACK response when a large message payload spans multiple network packets.

## Transmission.

Three message types in the transmission phase: the request, the simple reply, and the structured reply chunk.

e.g request:

## Request message

The request message, sent by the client, looks as follows:

C: 32 bits, 0x25609513, magic ( `NBD_REQUEST_MAGIC` )  
C: 16 bits, command flags  
C: 16 bits, type  
C: 64 bits, cookie  
C: 64 bits, offset (unsigned)  
C: 32 bits, length (unsigned)  
C: (*length* bytes of data if the request is of type `NBD_CMD_WRITE` )

e.g **simple reply** other than `NBD_CMD_READ` and if structured replies have not been negotiated :

S: 32 bits, 0x67446698, magic ( `NBD_SIMPLE_REPLY_MAGIC` ; used to be `NBD_REPLY_MAGIC` )  
S: 32 bits, error (MAY be zero)  
S: 64 bits, cookie  
S: (*length* bytes of data if the request is of type `NBD_CMD_READ` and *error* is zero)

e.g **structured chunk reply** :

A structured reply chunk message looks as follows:

S: 32 bits, 0x668e33ef, magic ( `NBD_STRUCTURED_REPLY_MAGIC` )  
S: 16 bits, flags  
S: 16 bits, type  
S: 64 bits, cookie  
S: 32 bits, length of payload (unsigned)  
S: *length* bytes of payload data (if *length* is nonzero)

## Terminating.

There are two methods of terminating the transmission phase:

- The client sends `NBD_CMD_DISC` whereupon the server **MUST** close down the TLS session (if one is running) and then close the TCP connection. This is referred to as 'initiating a soft disconnect'. Soft disconnects can only be initiated by the client.
- The client or the server drops the TCP session (in which case it **SHOULD** shut down the TLS session first). This is referred to as 'initiating a hard disconnect'.

## Magic values.

### Reserved Magic values

The following magic values are reserved and must not be used for future protocol extensions:

0x12560953 - Historic value for `NBD_REQUEST_MAGIC`, used until Linux 2.1.116pre2.

0x96744668 - Historic value for `NBD_REPLY_MAGIC`, used until Linux 2.1.116pre2.

0x25609514 - Used by nbd-server to store data log flags in the transaction log. Never sent from/to a client.

The following magic values are reserved and must be used only as described in the corresponding protocol extensions:

0x21e41c71 - `NBD_EXTENDED_REQUEST_MAGIC` Defined by the experimental `EXTENDED_HEADERS` extension.

0x6e8a278c - `NBD_EXTENDED_REPLY_MAGIC` Defined by the experimental `EXTENDED_HEADERS` extension.

## TLS Support.

Server decides to follow a TLS mode : NOTLS, FORCEDTLS, SELECTIVETLS  
Client

## Size constraints.

three size constraints: **minimum block, preferred block, and maximum payload.**

**If size constraints have not been advertised or agreed on externally, then a server **SHOULD** support a default minimum block size of 1, a preferred block size of  $2^{12}$  (4,096), and a maximum payload size that is at least  $2^{25}$  (33,554,432) (even if the export size is smaller); while a client desiring maximum interoperability **SHOULD** constrain its requests to a minimum block size of  $2^9$  (512), and limit `NBD_CMD_READ` and `NBD_CMD_WRITE` commands to a maximum payload size of  $2^{25}$  (33,554,432).**

A client MAY choose to operate as if tighter size constraints had been specified (for example, even when the server advertises the default minimum block size of 1, a client may safely use a minimum block size of  $2^9$  (512)).

### **The three size constraints in depth:**

The minimum block size represents the smallest addressable length and alignment within the export, although writing to an area that small may require the server to use a less-efficient read-modify-write action. If advertised, this value MUST be a power of 2, MUST NOT be larger than  $2^{16}$  (65,536), and MAY be as small as 1 for an export backed by a regular file, although the values of  $2^9$  (512) or  $2^{12}$  (4,096) are more typical for an export backed by a block device. If a server advertises a minimum block size, the advertised export size SHOULD be an integer multiple of that block size, since otherwise, the client would be unable to access the final few bytes of the export.

The preferred block size represents the minimum size at which aligned requests will have efficient I/O, avoiding behavior such as read-modify-write. If advertised, this MUST be a power of 2 at least as large as the maximum of the minimum block size and  $2^9$  (512), although larger values (such as 4,096, or even the minimum granularity of a hole) are more typical. The preferred block size MAY be larger than the export size, in which case the client is unable to utilize the preferred block size for that export. The server MAY advertise an export size that is not an integer multiple of the preferred block size.

The maximum payload size represents the maximum payload length that the server is willing to handle in one request from the client. If advertised, it MAY be something other than a power of 2, but MUST be at least as large as the preferred block size, and SHOULD be at least  $2^{20}$  (1,048,576) if the export is that large. Advertising a maximum payload size of 0xffffffff is permitted when the server does not have a fixed limit on client request payloads. Typically, the advertised maximum payload length is independent of the export size, even though the actual payloads for read and write cannot successfully exceed the constraints given by the export size and offset of a request. Notwithstanding any maximum payload size advertised, either the server or the client MAY initiate a hard disconnect if a payload length of either a request or a reply would be large enough to be deemed a denial of service attack; however, for maximum portability, any payload not exceeding  $2^{25}$  (33,554,432) bytes SHOULD NOT be considered a denial of service attack, even if that length is larger than the advertised maximum payload size.

## Metadata querying.

Metadata support is present by client queries ( Metadata querying section at <https://github.com/NetworkBlockDevice/nbd/blob/master/doc/proto.md#metadata-querying> ). May need this later.

## Values.

This section describes the value and meaning of constants (other than magic numbers) in the protocol.

When flags fields are specified, they are numbered in network byte order.

The link covers VALUES that have to do with: **Handshake phase, Transmission phase, Experimental extensions.**

<https://github.com/NetworkBlockDevice/nbd/blob/master/doc/proto.md#values>

## Compatibility and interoperability.

Originally, the NBD protocol was a fairly simple protocol with few options. While the basic protocol is still reasonably simple, a growing number of extensions has been implemented that may make the protocol description seem overwhelming at first.

This section of the proto file divides NBD in “optionals” and “musts” for people who want to implement the Protocol.

## Future considerations.

Structured replies; the Linux kernel currently does not yet implement them.

## End of proto.md file

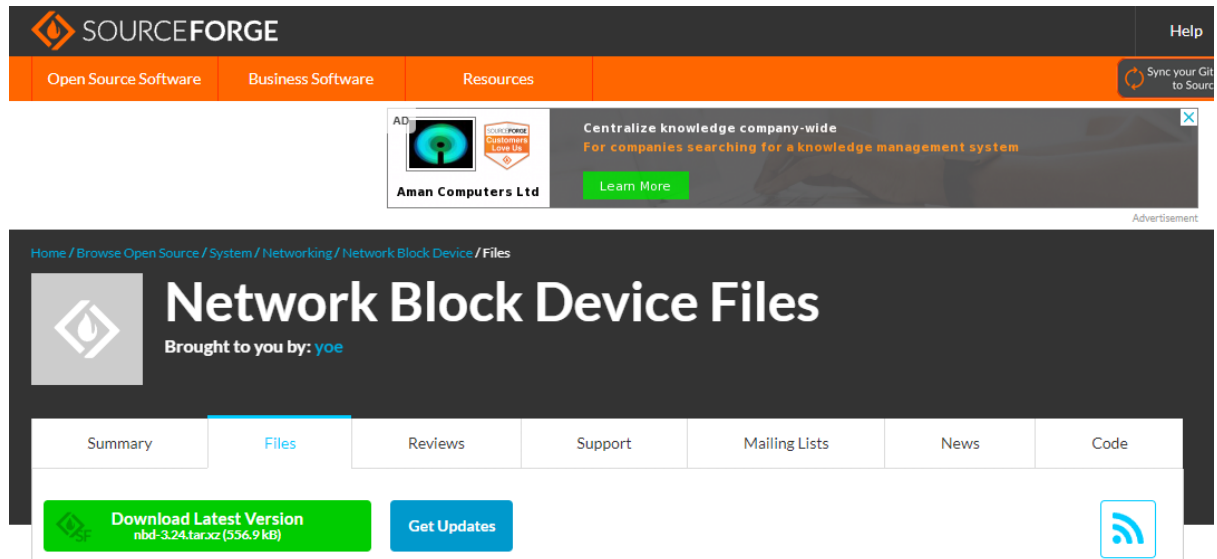
=====

## Start of README.md file

### NBD README.

This <https://github.com/NetworkBlockDevice/nbd> repo(package) contains nbd-server and nbd-client.

To install the package, download the source and do the normal configure/make/make install dance. You'll need to install it on both the client and the server. Note that released nbd tarballs are found on [sourceforge](https://sourceforge.net/projects/nbd/).



The screenshot shows the SourceForge website interface. At the top is the SourceForge logo and navigation links: Open Source Software, Business Software, Resources, and Help. Below this is an advertisement for Aman Computers Ltd. The main content area is for the 'Network Block Device Files' project, brought to you by 'yoe'. It features a breadcrumb trail: Home / Browse Open Source / System / Networking / Network Block Device / Files. Below the title are tabs for Summary, Files (selected), Reviews, Support, Mailing Lists, News, and Code. A green button labeled 'Download Latest Version' with a download icon and text 'nbd-3.24.tar.gz (556.9 kB)' is prominent. Next to it is a blue 'Get Updates' button. A RSS feed icon is also visible.

For compiling from git, do a checkout, install the SGML tools (docbook2man), and then run './autogen.sh' while inside your checkout. Then, see above.

## Using NBD.

First, on the client, you need to load the module and, if you're not using **udev**, to create the device nodes:

```
# modprobe nbd
# cd /dev
# ./MAKEDEV nbd0
```

(if you need more than one NBD device, repeat the above command for nbd1, nbd2, ...).

To start the **server**.

Next, write a configuration file for the server. An example looks like this:

```
# This is a comment
[generic]
# The [generic] section is required, even if nothing is specified
# there.
# When either of these options are specified, nbd-server drops
# privileges to the given user and group after opening ports, but
# _before_ opening files.
user = nbd
group = nbd
[export1]
exportname = /export/nbd/export1-file
authfile = /export/nbd/export1-authfile
timeout = 30
filesize = 10000000
readonly = false
multifile = false
copyonwrite = false
prerun = dd if=/dev/zero of=%s bs=1k count=500
postrun = rm -f %s
[otherexport]
exportname = /export/nbd/experiment
# The other options are all optional
```

The configuration file is parsed with GLib's GKeyFile, which parses key files as they are specified in the [Freedesktop.org Desktop Entry Specification](http://freedesktop.org/DesktopEntrySpecification), as can be found at <http://freedesktop.org/Standards/desktop-entry-spec>. While this format was not intended to be used for configuration files, the glib API is flexible enough for it to be used as such.

Now start the server:

```
nbd-server -C /path/to/configfile
```

The path is absolute.

To start the **client**.

```
nbd-client <hostname> -N <export name> <nbd device>
```

e.g.,

```
nbd-client 10.0.0.1 -N otherexport /dev/nbd0
```

**nbd-client** must be ran as **root**; the same is **not true** for **nbd-server**. (but do make sure that /var/run is writeable by the server that nbd-server runs as; otherwise, you won't get a PID file, though the server will keep running).

<https://github.com/NetworkBlockDevice/nbd#badges> here we can see the available packages for each corresponding OS



# End of README.md file



## Running TeraHeap with NBD

### Setup.

A cool setup way (on Ubuntu):

<https://sweetcode.io/introduction-to-linux-network-block-devices/>

Also:

<https://www.thegeekstuff.com/2009/02/nbd-tutorial-network-block-device-jumpstart-guide/>