Networking the QL and ZX Spectrum – Part 1

Martyn Hill, v2.2 - Jan 2025

Introduction

This set of three articles should prove of interest to any QL or ZX Spectrum user wishing to connect their QL/Spectrum compatible machines to one another using the in-built network capability, to share files or investigate the potential for multi-user gaming, or else just fascinated with this simple but surprisingly effective network technology.

The original version of this article was focused on the QL but as there is significant and intentional overlap between the networking of these two machines, the article has since been extended to cover the Spectrum/Interface-1 as well. When referencing capabilities specific to either platform, the terms **QLAN** (QL) and **ZX Net** (Spectrum) will be used to differentiate.

Furthermore, in this latest version (Dec 2024) the recent implementation of the network for the ***ZX Spectrum Next*** when running the new '**QL-Core**' by Theodoulos Liontakis has also been added. In due course, a native Spectrum Next network driver will be developed, and this article will be updated accordingly.

During my exploration of the Sinclair Network over several years I have come to appreciate the simplicity and elegance of its design, and I would like to acknowledge the ingenuity of its original designers, Sinclair Research Ltd and Tony Tebby. I must also thank QView (and Lau Reeves, in particular) for the meticulous annotation of the Minerva source-code that aided my early understanding of the QLAN network driver, before I then embarked on making-sense of Tebby's excellent enhancements to QLAN in Toolkit-2 (TK2).

A similar investigation was made into the Shadow ROM Network code as built-in to the ZX Interface-1, in part driven by the reported incompatibilities between the QL and Spectrum implementations of the network, which are now well understood and details of how to effectively mitigate these issues are presented later.

Whilst researching the various alternative Spectrum ROMs, Geoff Wearmouth's 'Sea Change' ROM (see http://zxspectrum.it.omegahg.com/rom/seachange/seachange.pdf) commentary suggested that this network protocol was termed the ***Sinclair Network Standard*** and may have been envisaged as a more universal standard – though, to-date, implementations exist only for the QL (and its clones) as well as for the ZX Spectrum computer with the ZX Interface-1 attached. *As the original MGT Disciple Disc interface was also equipped with the same network hardware and Shadow ROM routines, any references to the Interface-1 can be assumed to include the MGT interface.*

About the author…

I work within the Cybersecurity industry, assisting customers realise the full value of their real-time threat detection and remediation solutions within the Security Operations Centre (SOC). Interwoven with my IT career, I also spent 7 years in education, teaching ICT and running the IT department for a secondary school.

I live in the UK with my small but wonderful family and spend some of my free-time developing retro computer projects with an emphasis on computer networking – this set of articles is a culmination of much of my research and experience to-date on this topic.

The subject of heterogeneous computer networking and interfacing diverse electronic and computing devices has always held a fascination for me and I have been experimenting with the QL Network specifically since I acquired a second-hand QL from a generous colleague about 10 years ago. This was many years after having owned my *first* QL as a teenager, thanks to my brother who worked at the time for Thorn EMI Datatech, contracted by Sinclair to service the QL and Spectrum

hardware. It was also he who gifted-me my first ever home-computer, a ZX Spectrum, at the age of 11 and undoubtedly influenced my subsequent career choices – thanks, brother!

Before re-acquiring a real QL, I had used various QL emulators running on my PC to develop software and ideas, finally settling on the terrific QPC2 by Marcel Kilgus, which I continue to use as my main QDOS platform – but I was missing the opportunity for hardware development that is so much more accessible with the original QL hardware than with more modern computing platforms.

I now own 4 QLs of various states of expansion (one with Tetroid's SGC clone, fitted inside a PC case), a QXL card also installed in the same PC, a couple of Peter Graf's marvelous FPGA-based Q68s as well as four ZX Spectrum/Interface-1s and, more recently, a ZX Spectrum Next - all of which I have connected together successfully using the QL/ZX Network. I don't classify myself as a 'collector' – any earnest retro computer buff would be horrified to see what I've done to these vintage computers – and the proliferation of QLs and Spectrums in my home-lab was driven solely by the desire to test and compare the behavoiur of the network between different versions and builds of these machines – much to the consternation of my ever-patient wife…

Contents

The following topics will be covered in this set of articles:

1. Introduction to the Sinclair Network, its capabilities and compatible systems *(this article)*
    - *Useful applications/use-cases for connecting QL and Spectrums*
    - *Typical problems and troubleshooting*
    - *Extending QLAN with TK2 on the QL*
    - *Specific considerations when connecting a QL to the ZX Spectrum*
    - *Connecting the ZX Spectrum Next/QL-Core*

2. A technical deep dive into the QLAN/ZX Net protocol and the hardware that it runs over
    - *The basic QL NET device in hardware and software*
    - *The equivalent 'N' device on the Spectrum/Interface-1*
    - *How TK2's FSERVE extends these capabilities for the QL*
    - *How the network protocol looks 'over-the-wire'*

3. Ideas for the future of networking using the Sinclair Network Standard
    - *Enhancing the protocol whilst preserving 'down-level' compatibility*
    - *Leveraging additional hardware to overcome the main limitations*
    - *Extending the Sinclair Network to other retro machines of the era*
    - *Bridging with other network stacks*

Other Resources

There are some excellent resources describing QLAN/ZX Net already available online – notably:

- Roy Wood/Q-Branch's site **SuperBASIC/SBASIC Reference Manual Online – Section 17** (https://superbasic-manual.readthedocs.io/en/latest/index.html) - with content from F Herrmann, P Jager, R Mellor and N Dunbar
- An article entitled **Network** by David Denham originally published in QL Today, available on Dilwyn Jones' site: http://www.dilwyn.me.uk/docs/articles/network.zip
- The original **Sinclair QL User Guide**, also available on Dilwyn's site: http://www.dilwyn.me.uk/docs/ebooks/olqlug/index.htm)
- **The Spectrum Microdrive Book** by Dr Ian Logon is another valuable resource for anyone wishing to better understand the ZX Net implementation

- **Master Your ZX Microdrive** by Andrew Pennel – another excellent resource (as ever, from Mr Pennel)
- **Spectrum Shadow ROM Disassembly** by Gianluca Carri

Inevitably, there is some duplication below of information already covered elsewhere, but the aim of this article was to focus on what is *not* already documented, as well as to clarify – and in some cases, to correct – previously documented information. The hope is to share something of my own fascination in this area and, with any luck, present something *useful* that you might not have read before and, ultimately, to encourage further adoption and even development of new ideas for the Sinclair Network!

Alternative Network solutions

Before describing the Sinclair Network, it should be noted that there are several alternative methods and technologies available to inter-connect the QL and Spectrum, each with their own strengths and some intended to extend the local LAN to the Internet. However, a conscious decision was made when writing these articles to focus on the *Sinclair Network Standard* native to *any* QL or Spectrum/Interface-1.

Here is a non-authoritative list of some of the alternative networking solutions known to the author and the reader is encouraged to research any that are of interest:

- The **IPNet** driver by Martin Head
- Xelalex's very capable **ZX Net with OqtaDrive**
- Emerging or well-established IP-based networking solutions such as **FujiNet** or **Acorn's AUN** (EcoNET over IP)
- The **SERNET** solution for the QL (originally developed by Bernd Reinhardt and based on Phil Borman's MidiNet software
- The **WiFi Modem** project to add wireless IP capability via the QL's serial port

There are several other retro network protocol stacks that, with some thought and effort, could be re-implemented on the Sinclair machines and which were reviewed carefully by the author during the preparation of this article, the most notable being Acorn's **original EcoNET** (with various levels of Fileserver) and Apple's **AppleTalk stack**. We may touch on these in a later article.

Of special interest is Martin Head's **QLAN over Ethernet** driver, currently implemented on the Q68 FPGA-based QL with its in-built NIC as well as for PC-based QL emulators and which the author uses successfully alongside the native Sinclair Network to successfully 'bridge' between distinct network segments in his home lab, using a pair of Q68's, one on each 'end' of the bridge. Mention will be made of this approach later.

Modern Driver Development

During my exploration of the networking capabilities of these Sinclair machines, several enhanced versions of the QLAN device-drivers have been developed across the various QL platforms as well as the fixing of previously undocumented bugs in the network code of the Spectrum/Interface-1 Shadow-ROM.

Using these enhanced drivers, it has been possible to run the Sinclair Network at anything from 4.5x to 8x the original bit-rate with suitably swift QL-like hardware.

A hardware/software solution has also been developed and openly published to allow a QL-emulator running on a PC to connect to the Sinclair Network via the host-PC's USB port – named the 'QLUB Adapter', as well as the porting of Tebby's QL network code to the Q68 (and QZero) FPGA-based QL – a project named 'ND-Q68'.

With the arrival of the ZX Spectrum Next, the Q68 NET driver has recently been ported to the Next's **QL-Core** – similarly named 'ND-Nxt.'

Thanks to the clear design and copious source-code comments, it was relatively straight-forward to take Tebby's original TK2 network driver from the SMSQ/E source-tree and re-engineer it for the Q68 to utilize its hardware timer/counter and, in the process, remove the dependency on m68k and Z80 instruction-cycle timings which dictate the design of the original NET device drivers – the principal cause of communication problems between different machines.

All these drivers have been made available online and will continue to be maintained and shared openly with the community - with thanks to the original developers, Sinclair Research and Tebby.

1. Introduction to the Sinclair Network

*It is recommended that you read this section in conjunction with the **SuperBasic/SBASIC Reference Manual Online** – section 17 'Networking'. I have consciously omitted details that are already well-documented there, duplicating only what is needed to contextualise any additions to the subject, or to correct some small errors found in that otherwise very comprehensive resource.*

1.1. QLAN/ZX Net Compatible Systems

In addition to the basic QL, the **Aurora** QL mainboard replacement, the **QXL** PC ISA card and the newer **Q68** FPGA-based machine all include all – or most - of the hardware needed to interconnect; the Q68 will require a few additional components to be retro-fitted (see the Sinclair QL Forum posting at https://www.qlforum.co.uk/viewtopic.php?f=3&t=2881)

A QL fitted with a **Gold** or **SuperGold Card** will of course run QLAN very nicely already – all the required protocol timings being automatically adjusted in software to suit their faster CPUs.

The **CST Thor** range of machines also have the required QLAN hardware and software.

In addition to QL-compatible machines, functionally-similar hardware and the same basic protocol is used by the **ZX Spectrum** when the **Interface-1** is fitted but, due to some subtle bugs in the Interface-1 Shadow ROM, reliable reception at the QL *from* the Spectrum can only take place with a updated Interface-1 firmware or alternatively, with a slightly optimised QL NET driver. This is a real pity, as many users transitioned to a QL from the Spectrum and might have benefited from connecting the two, otherwise incompatible machines, without recourse to the slower serial-ports.

The **ZX Spectrum Next** can already accept and run the Interface-1 when in native Spectrum 48/128 mode, and now with the ND-Nxt solution, the Next/**QL-Core** can be connected to the Sinclair Network using only a simple adapter connected to the joystick port (the 'Joy2NET' adapter) – economically and easily constructed by anyone with some soldering or breadboarding skills. *A suitable schematic for this adapter is presented in the Appendix.*

Other *potential* candidates for connection with the Sinclair Network include the **Q40/Q60**, **SAM Coupe** and **Enterprise ELAN** but these machines would require additional (but relatively simple) hardware interfacing and suitable software to be developed.

1.2. Typical Applications for QLAN/ZX Net

Simple file-transfer between two or more stations is perfectly possible with the basic Sinclair Network without the enhancements afforded by TK2 (covered later).

The network can be put to good use in this simple **Peer to Peer** mode, e.g.:

a) **'Boot-strapping'** – transferring S/Basic extensions and applications to an unexpanded QL/Spectrum at start-up.
b) **Preserving your work** – saving in-memory files to another working machine when you can't get the local microdrives to cooperate.
c) **'Spooling' print files** – using COPY to another machine with an attached printer to get hard-copy.
d) **Multi-player gaming** – streaming real-time game data between two or more machines and players. A limited range of network-supported games are available for both the Spectrum and QL – *we need more!*
e) **Cross-platform computing** – whilst binary code and Basic files are not directly compatible between QL and Spectrum, once the different system architectures are accommodated, the availability of a real - or near real - time connection affords the opportunity to develop computing projects across these diverse platforms, leveraging the strengths of each.
f) **Electronic-interfacing projects** – with a suitably programmed microcontroller to speak the Sinclair Network Standard, it becomes possible to interface the QL/Spectrum with your electronic breadboarding projects (more in a later article.)

Once a working network is established, timing and reliability of the network can actually outperform access of ageing MDV cartridges, even though the *raw* bit-rate of the network (c. 90kbps) is much lower than that for microdrives (200kbps nom.) – EXECing large programs over the network can often prove more reliable *and* more rapid than loading from a flaky cartridge, where multiple passes of the tape are often required to correctly read the file-blocks.

Given the interactive nature of the NET device effectively requiring commands to be issued on *both* peer stations simultaneously before file-transfer can begin, use of the basic NET functionality is not especially convenient unless both keyboards are accessible, but effective none-the-less. As we'll read later, TK2 addresses this inconvenience very effectively on the QL with its FSERVE file-server capability…

1.3. Connecting Stations

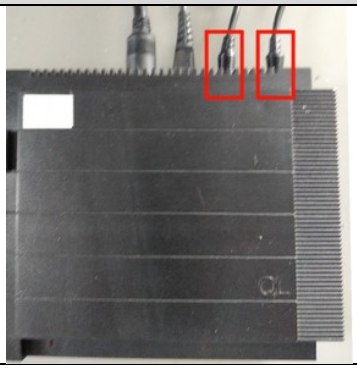
Every QL and Spectrum/Interface-1 - and several of their compatible hardware replacements – include the basic hardware and software to allow connection to one another using a simple two-wire cable, such as that used for mono-audio applications, terminated with a 3.5mm 'jack' plug of the **Tip-Sleeve** (**TS**) varieties at each end.

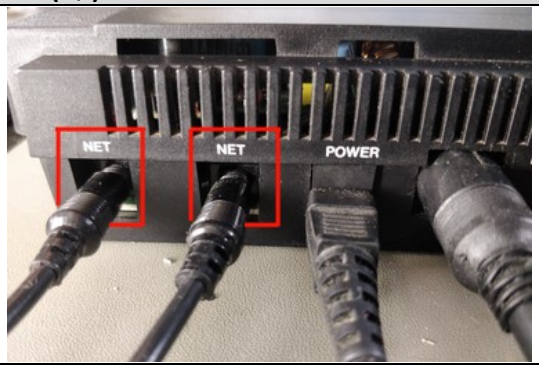
It is also perfectly acceptable to use 3-wire, stereo cabling with the **TRS** (Tip-Ring-Sleeve) type plug.

NET Port Connection Plugs	
	
3.5mm 'TS' Jack-plug (mono)	**3.5mm 'TRS' Jack-plug (stereo)**

Each station is linked to the next in a daisy-chained/common-bus arrangement using either or both of the two (identical) NET sockets that lurk unobtrusively at the rear of your QL and Interface-1; the two stations that sit at the extreme ends of the chain ***will only have one socket connected***, whereas each intervening station will have both occupied.

NET Port Sockets (QL)

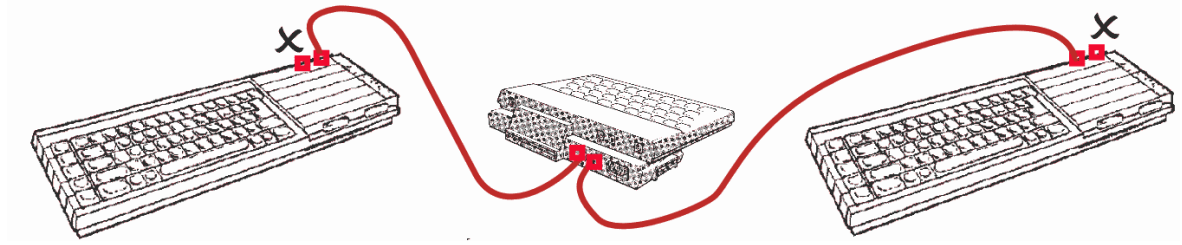




The most common situation of just two connected stations will therefore require only a **single** 2-core cable, leaving one NET socket disconnected on each station – it does not matter which of the two available sockets are used at each end.

Leaving the two end-stations with a single unoccupied socket on each is actually part of the network hardware design – the sockets are themselves of the 'switched' type, whereby any unconnected socket provides a pull-down termination to the NET line; once a jack-plug is inserted, the termination is effectively disabled on that particular socket.

Sinclair Network Cabling



QL image taken from the QL User Guide
Spectrum/Interface-1 image taken from the Microdrive and Interface-1 Manual

Inadvertently creating a loop with the two end-stations also linked to one another will thus cause poor reception due to 'signal reflections' and is likely to render the network unreliable.

The simplicity, availability, low cost and robust nature of the required cabling is, in the author's opinion, a key benefit of the Sinclair Network design over, say a traditional RS-232/COM port solution, or even Ethernet cabling.

The maximum *combined* length for the cable-run is stated to be 100m (QL User Guide, p34.) In practice, the author has seen success using cable-runs of 30m or more between individual stations using off-the-shelf and inexpensive mono and stereo-audio cabling.

1.4. Basic Network Usage (Peer to Peer)

The basic NET device driver built-in to the QL's QDOS allows for simple, **peer-to-peer** transfer of files using familiar S/Basic commands such as `COPY` and `LOAD`, `LBYTES`, `EXEC` (with their `SAVE` counterparts), as well as sending/receiving arbitrary byte-streams using explicit `OPEN/PRINT` and `OPEN/INPUT/INKEY$` command sequences at the respective ends of the link.

Similarly on the Spectrum (assume '*with Interface-1 attached*' hereafter), the extended Shadow ROM commands and syntax apply as usual, using the 'N' device specifier.

Each station is assigned a unique 'station-ID' between 1 and 63 using the S/Basic `NET` command or the Spectrum's `FORMAT "n";x` syntax - and a file-transfer is initiated by entering reciprocal commands (load/save) at each station.

If only two machines are connected together, it is possible to successfully communicate *without* changing the station-ID from its default of '1' – ambiguity is avoided by the fact that only one activity (send *or* receive) can take place at any given time on either station. That said, assigning unique station-IDs is recommended even in this situation.

In addition to specifying the remote peer station-ID when opening a channel, the NET/N-device also recognises a station-ID of `0` (zero) for both input and output - referred-to as a '**network broadcast**.'

In addition, QLAN supports a 'receive from any station' facility, whereby the *receiving* station NET channel is opened to *its own* station-ID, ready to accept connections from an initially unknown station. Once such a connection is established, QLAN will continue to receive subsequent packets only from the station with which it initially connected.

The basic Sinclair Network protocol detects the end-of-file (**EOF**) condition to determine when to close byte-stream type connections - as created with an explicit `OPEN`; the NET driver flags the last packet as such when the output channel is `CLOSE`d, suiting 'byte-serial' communications.

However, this method is insufficient for file-transfer type connections initiated with `SAVE`/`SBYTES`/`SEXEC` and received with `LOAD`, etc, where it is necessary for the receiving station to know *from the outset* the expected file-length and its type. To accommodate this, the file-transfer commands (save, etc) **prepend** a 15-byte 'file-header' (QL) or a 9-byte header (Spectrum) to the very first packet sent to indicate the number of bytes to follow, along with other meta-data – which is expected and interpreted by the receiving end. This use of file-headers will be recognisable to readers familiar with the ZX Spectrum's 17-byte tape file-header (the Interface-1 equivalent file-header excludes the 10-bytes of filename.)

Some example S/Basic (QL) routines might help clarify:

Transferring Data and Files between QLs	
QL #1	***QL #2***
Stream serial data	

`NET 1`
`OPEN #3,'neto_2'`
`FOR b%=32 TO 192`
`  PRINT #3, CHR$(b%);`
`END FOR b%`
`:`
`CLOSE #3`
	`NET 2`
	`OPEN #3,'neti_1'`
	`REPeat recvBytes`
	`  IF EOF(#3) THEN EXIT recvBytes`
	`  PRINT INKEY$(#3,-1);`
	`END REPeat recvBytes`
	`CLOSE #3`

Transfer a file (e.g. QL screen-display)	

`NET 1`
`REMark Implicit file-channel automatically opened for output, then closed`
`:`
`SBYTES 'neto_2',131072, 32768`
`:`
`REMark ...Or from a saved file`
`COPY 'win1_test_scr' TO 'neto_2'`
	`NET 2`
	`REMark Implicit file-channel automatically opened for input, then closed`
	`:`
	`LBYTES 'neti_1',131072`
	`:`
	`REMark ...Or, save it to a file`
	`COPY 'neti_1' TO 'win1_test_scr'`

A simple QL/Spectrum example - *for more detail, see section 1.8 "**Connecting with the ZX Spectrum**"*

Transferring Data between a QL and Spectrum	
QL-A	***Spectrum-B***

`NET 1`
 	
`COPY 'neti_2' TO 'win1_test_scrZX'`
 	
`COPY_N 'win1_test_scrZX' TO 'neto_2'`
	`FORMAT "n";2`
	`  Load something to the display-file,`
	`  then…`
	`SAVE *"n";1 SCREEN$`
	
	`CLS`
	`LOAD *"n";1 SCREEN$`

NB: In the above example, as we do **not** want a QL-style header to be added during the transfer *back* to the Spectrum, we use the `COPY_N` command variant - the file stored on the QL as `test_scrZX` will already include the raw 6,912 bytes in the standard Spectrum display-file format *pre-pended* with a *9-byte file-header* when it was SAVEed from the Spectrum.

Another example, providing a 'DHCP' like facility to automatically assign NET station IDs:

Example 'NET Station-ID Server' (QL)

Server (QL)

```
idServer%=32: nextId%=2: maxId%=31: keyEsc%=27
:
NET idServer%
:
REPeat serveNetIds
  REMark *** (re)Open an input channel to self ***
  OPEN #3,'neti_'&idServer%
  REPeat waitClient
    tempClientId%=CODE(INKEY$(#3,10))
    IF tempClientId%<>0 THEN EXIT waitClient
    IF CODE(INKEY$(#0,10))=keyEsc% THEN EXIT serveNetIDs
  END REPeat waitClient
  :
  OPEN #3,'neto_'&tempClientId%: PRINT #3,CODE(nextId%);: CLOSE #3
  nextId%=nextId%+1: IF nextId%>maxId% THEN EXIT serveNetIds
END REPeat serveNetIds
:
CLOSE #3
```

A suitable client-side program could be (both for QL and Spectrum):

Client (QL)

```
idServer%=32
REMark *** Generate a random but 'reserved' temporary station-id
netId%=RND(33 TO 63)
:
NET netId%
:
OPEN #3,'neto_'&idServer%: PRINT #3,CODE(netId%);: CLOSE #3
:
OPEN #3,'neti_'&idServer%
getId%=CODE(INKEY$(#3,5*50))
CLOSE #3
:
IF getId%<>0 THEN
  NET getId%
ELSE
  PRINT #0,"Couldn't get NET ID. Quitting...": NET 1: STOP
END IF
```

Client (Spectrum)	idServer=32 REMark *** Generate a random but 'reserved' temporary station-id netId=33+(RND * 64) : FORMAT "n";**netId** : OPEN #3,"n";**idServer**: PRINT #3,CODE netId;: CLOSE #3 : OPEN #3,"n";**idServer** <getIdlabel> getId=CODE INKEY$ #3: IF getId=0 THEN GO TO <getIdlabel> CLOSE #3 : IF getId<>0 THEN FORMAT "n"; **getId**: STOP : PRINT "Couldn't get NET ID. Quitting...": FORMAT "n"; 1: STOP

The current NET transmission can also be aborted manually at any time from either end by the user pressing `Ctrl+Space` (QL), or `CAPS+Space` (Spectrum) triggering the closure of any *implicit* file-transfer channels in the process. An 'EOF' condition will be flagged and detectable at the receiving-end if aborted at the sender station. However, the sender has no way to know if the user at the receiving station has since aborted the transfer and will instead continue to re-attempt transmission of the latest packet unless and until it is manually aborted itself.

1.5. Packet Anatomy

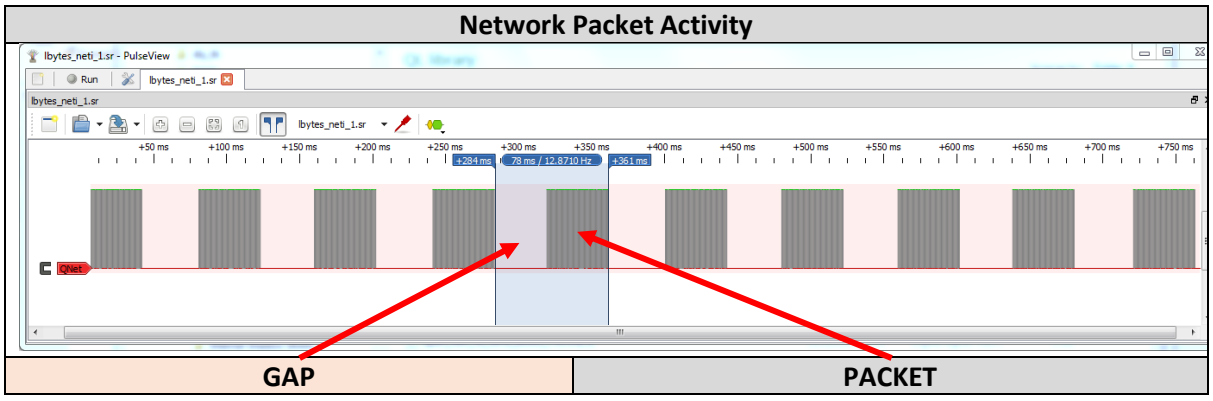
It may prove interesting to understand how the Sinclair Network protocol arranges for the raw data to appear over the link, how it manages time for other activities in the machine as well as sharing network access-time with other connected stations. This topic will be further expanded in a later article.

All transmitted data is split in to 'blocks' or packets of up to 255-bytes, prepended by a packet-header followed by the data-packet itself. For a file consisting of multiple blocks then, all but the last will be the full 255-bytes in length, with the last containing the remaining bytes.

Note that the packet or block header is distinct from any file-header that may appear in first of the actual data packets.

The packet-header includes both the **source** and **destination** station IDs (as a pair of 8-bit octets), as well as other meta-data used by the protocol including the current **block-number** (16-bit word) and checksums (see https://superbasic-manual.readthedocs.io/en/latest/Appendices/Appendix17.html#a17-1-4-qnet-without-toolkit-ii)

A sample signal trace showing several packets of an on-going transmission shows the general link-usage:



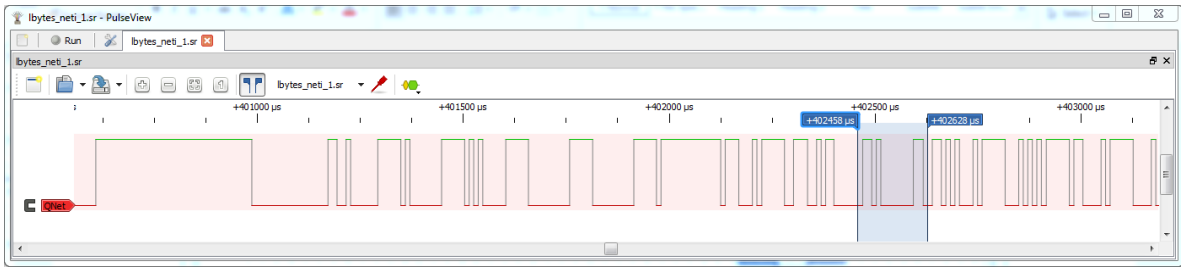
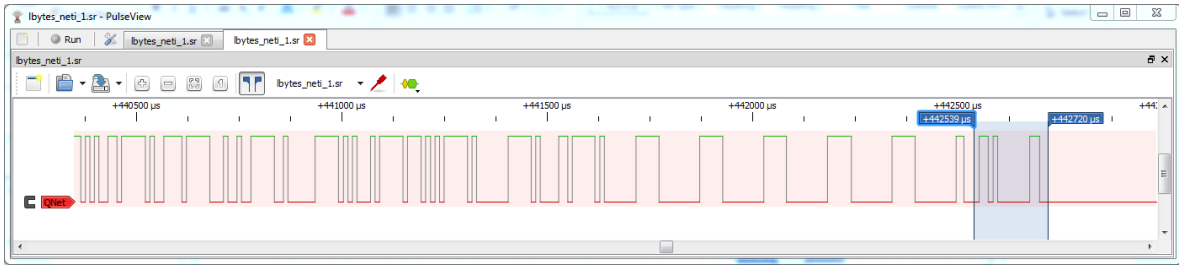
The sending station starts by 'listening' for a suitably long silence or **gap** indicating that the link is free, before placing a **scout** bit-pattern on the wire whilst simultaneously reading back the link state to ensure that no other station is also trying to claim the link.

The bit-pattern used for the scout is carefully computed both to ensure that different sending stations produce a **distinct pattern** (based principally on their own, unique station-ID) as well as to ensure that a **predictable outcome** will result. The predictable outcome in this case is such that all stations *except* the one with the *lowest* station-ID will detect the **line-contention** and back-off gracefully, whilst the sending station with the lowest station-ID will not even notice the contention and proceed with its transmission. Simple, but effective…

At the receiving end, once the *start* of the scout is detected, the receiving station actually *ignores* the link state altogether, only starting to pay attention again after a defined time-window has passed designed to skip the scout-phase, but in good time to catch the start of the packet-header that follows immediately.

Following transmission of the packet-header, the sending station listens for an active 'acknowledge' signal (a single byte: 0x01) to be received-back before initiating transmission of the data-packet itself and, once again, listens for a final acknowledgement to determine that the **data-packet** was successfully received. If successful, the block-number is incremented at each end, ready for the next packet to be transmitted.

Zooming-in to the signal trace during the packet transmission shows the following:

<table>
<tr><th colspan="6">Sinclair Network Packet Detail</th></tr>
<tr><td colspan="6"> 
</td></tr>
<tr><td>Gap</td><td>SCOUT</td><td colspan="2">HEADER</td><td>H-ACK</td><td>DATA >>></td></tr>
<tr><td colspan="6"> 
</td></tr>
<tr><td colspan="4">>>> DATA</td><td>D-ACK</td><td></td></tr>
<tr><td></td><td colspan="2">Sending Station</td><td></td><td colspan="2">Receiving Station</td></tr>
</table>

The sending station will **abort** the current transmission-cycle if any of the following conditions are detected:

- A gap of sufficient length is not detected – *indicating that the link is already in use*
- Contention found when sending the scout - *another station is simultaneously trying to claim the link*
- Either acknowledge response-byte (header or data-block) is not received - *receiver not listening, or has rejected the packet for some reason*

On the QL, the QDOS IOSS scheduler will attempt a re-transmission of the current block later. On the Spectrum with its single-threaded OS, transmission is instead re-tried almost immediately – only after a slightly randomized delay and checking the keyboard for a user pressing break. In either case, the current block-number is left alone at each end.

The receiving station - which is periodically 'polling' the link for the expected block/packet – will **abort** this poll-cycle if:

- It fails to detect the scout within a defined time-window
- After reading and inspecting the packet-header, determines that the packet is not meant for, or currently expected by, this station
- Calculates a corrupted check-sum in either of the header or data-packets

The receiver will respond by simply *not* sending a response byte (header or data-block, as appropriate) and, like transmission, on the QL rely on the QDOS IOSS scheduler to re-invoke packet-reception later, or retry after a randomised period of time on the Spectrum.

Alternatively, if the header and data-blocks are received **successfully**, both the receiver and the sender adjust their respective copy of the current **block-number** by ***incrementing*** it, ready for the next packet/block and then passing control back to the OS to process the latest data-block.

Thus, the sender and receiver stay in-sync by referring to the incremental block-number **N** (0-65535) - such that any repeated packets (block **N−1**) can be detected and discarded at the receiver, whilst continuing to wait for the expected packet (block **N**). Such a situation can arise when the sending station fails to recognise the acknowledge-byte sent in response to the last packet sent.

When the sender later attempts re-transmission of the 'repeat' packet, it has to be discarded by the receiver **but** *still acknowledged*, so that the sender finally knows to 'move-on' to the next block.

One of the bugs found in the Spectrum Interface-1 Shadow ROM relates to this process, whereby the receiving Spectrum would fail to correctly detect a repeat packet (and thus not acknowledge it), leaving the sending station trying to resend an old packet in an infinite loop.

Network 'broadcast' transmissions are much the same **except** in how acknowledgements are handled (and the length of the Scout-phase.) On a QL without TK2 and on the Spectrum, *no* acknowledgement handshaking is employed leading to unreliable broadcasting.

With TK2 on the QL however, an **active ACK**/**Negative ACK** is added to the protocol, transmitted only after the end of the data-packet (*not* after the header) that improves broadcast reliability immensely.

The length of the Scout during a broadcast from a QL with TK2 is also extended by 5ms to allow time for the receiving QL (assumed to also be a QL with TK2) to poll the keyboard – a rather lengthy process on a standard QL.

Note however that these enhancements effectively render network broadcasts ***incompatible*** between a QL *without* TK2 (or Spectrum) and a QL *with* TK2.

1.5.1. Mind the Gap…

Gaps are seen interspersed between each attempt to transmit a packet, both between successful and unsuccessful transmissions; the length of the gap between *failed* attempts is typically shorter.

These transmission gaps may seem wasteful on the effective link-utilisation but actually form an integral part of the protocol - allowing time for the receiver to process the last packet and the sender to prepare the next, as well as to free the line long enough for *another* pair of stations to claim the link during the intermission, effectively time-sharing the half-duplex bus.

This approach to time-sharing a link and listening for contention (during the scout phase) is similar to what is referred-to as 'Carrier Sense/Collision Detection, Multiple Access' or 'CS/CDMA' in conventional network terminology.

To give an indication of the impact of these gaps on effective throughput, we can observe that, between two 7.5MHz QLs with no other jobs running and no corrupted packets, a 32KB QL display-

file will transfer in about **10.5 seconds**, with about **45%** of this time spent in the gaps between actual data packets.

Between two Q68 machines - or the Spectrum Next/QL-Core - on the other hand, the same file is transmitted in about **6.3 seconds** with only **15%** of the total time spent in the gaps, due in part to the more rapid inter-packet processing, before each station is then ready for the next.

This evaluates to transfer-rates of between 3KB and 5KB per second, depending upon the performance of the machines involved and the minimum length of gap that each can sustain. Thus, higher effective throughput is achieved with an improved link-utilization/efficiency for the very same bit-rate.

If this time spent in gaps seems excessive, bear in mind that, for an inherently multi-tasking OS such as QDOS, the transmission link gaps *also* represent time-slices now available to the CPU to devote to other tasks/Jobs that may be ready to run and thus maintain a higher level of overall responsiveness to the user.

The Spectrum ROM, on the other hand, being inherently mono-tasking, typically uses shorter gaps in its transmission attempts – but only scans the keyboard for 'Break' between each packet with no other user activity possible until the expected packet is successfully received and processed.

In summary, the QLAN and ZX Net protocols are based on a packet re-transmit mechanism (much like Ethernet), whereby it is entirely acceptable for individual packets to be 'missed' when the receiver is not currently listening, and continually re-tried until an active acknowledgement is received and the next packet can be queued for transmission. The IOSS 'retry' mechanism in QDOS is therefore an essential support to the QLAN protocol within its multi-tasking environment.

The 'attentiveness' of the receiver (i.e. the frequency of polling) is another factor in the effective throughput – more so than the raw network bit-rate - and one reason why a machine like the Q68 and Next/QL-Core achieve a better *reception* throughput than the basic QL or Spectrum.

1.6. Trouble-shooting Network problems

Many QL users have tinkered with the network, only to get frustrated and give-up without ever enjoying the benefits of networking. This is especially true when trying to link QLs and Spectrums and there are several potential reasons for this, the most common of which are described below in terms of either their cause or the observed behaviour. *Possible cures* are provided in each case:

1. **Unreliable QL hardware** – the earlier issue-5 QL motherboard has the ZX8302 ULA responsible for the network and microdrives connected to the 'contended' data-bus of the QL's main ULA, resulting in irregular access-timing which can cripple QLAN. Sometimes they work, another day, not. *Replacing with an Issue-6/7 QL motherboard helps to resolve this.*
2. **Stuck NET ports** – it is occasionally observed that the voltage present at the NET port gets 'stuck' at one logic-level or the other, indicative of either a failing PNP transistor in the NET output circuit (QL or Spectrum), or a defective/marginal ZX8302 ULA (QL). *Power-cycling the responsible machine can often 'release' the stuck NET port. Replacing the transistor (TR2/ZTX-510 or equivalent) can also help.*
3. **Corroded NET sockets** – given their typical lack of use, the inner contacts of the NET jack-sockets can become tarnished or even rusted, causing poor or no connection. *An attempt to clean the inner contacts with Isopropyl alcohol and lint-free swab or similar may help (see a user's suggestion at* https://qlforum.co.uk/viewtopic.php?f=12&t=3279*), or else de-soldering the sockets and replacing with a similar variety (if you can still find them) if truly corroded. Remember that the two sockets are ganged together, so the state of one will impact the other, so treat them as a pair.*
4. **Faulty cabling** – if you have made your own – or else re-purposed an old 2-core cable with jack-plugs, the connection failure may be in the soldered plug or within the cable itself.

Check continuity and that the respective 'poles' of the jack-plug have been connected to their counterpart at each end – Tip to Tip, Sleeve to Sleeve.

5. **Mis-connected cabling** – with more than a couple of machines linked together, it is surprisingly easy to mess-up the cabling, resulting in one or other station being left disconnected, routed back to itself, or creating an unwanted 'loop' between the two extreme ends of the link, thus disabling the line-termination. *Recheck the cabling – it is best to abort any on-going transmission at the sending station before unplugging/replugging cables.*
6. **Misconfigured Station IDs** – a simple, but common enough occurrence, again when more than two machines are connected, is forgetting to use the `NET` (or `FORMAT "n"`) command to set each machine with an unique station-ID. Alternatively, forgetting the station-ID of the machine you are typing-at when entering the respective target/destination device name. *Check and re-issue the `NET/FORMAT "n"` command or re-enter the appropriate device name.*
7. **Broadcast failure** – the station-0 network-broadcast feature (using `NETI_0`/`NETO_0` or `"N";0`) is not especially reliable due to the lack of active 'acknowledgement' in the broadcast protocol. *Test again using a direct peer-to-peer file-transfer to first validate the link, then retry the broadcast. Upgrading a legacy QDOS ROM to the marvellous Minerva OS can also make network broadcast more reliable due to the re-coding of this feature in Minerva's NET driver – or add TK2, which replaces the broadcast feature entirely with a much enhanced version.*
8. **QL seems to hang** – this may not be a fault, but a normal part of using the network as interrupts are switched-off within the NET 'physical' device-driver during transmission of each packet to ensure consistent timing. In particular, if you abort a transmission in progress, the sending station will repeatedly retry transmitting the current packet up to 2,000x, paying *minimal* attention to the keyboard for the user pressing `Ctrl+Space` between each attempt. The receiving station is usually more responsive to `Ctrl+Space`.
9. **Nothing is received when 'streaming' serial data** – again, this may not be a fault when using an explicit `OPEN`, or even `COPY`ing from another 'serial' type device. The packet/block design of the NET driver means that a full packet of 255 bytes needs to accumulate in the sender's buffer before the physical driver is invoked to transmit down the wire. *As there is no 'FLUSH' capability in the current NET driver, it is necessary to either pad-out the output to fill the 255-byte buffer, or else issue a `CLOSE` at the sending station. When closed, the NET driver will attempt to send whatever is present in the buffer regardless of how full, marking this the 'last' packet/block. In turn, this will also cause the receiving station to detect an EOF condition.*
10. **Bad Parameter (QL) or Wrong File Type (Spectrum) error** – when receiving a file using `LBYTES`/`EXEC` etc on the QL, or `LOAD` on the Spectrum, the receiving station expects a simple 'serial file-header' as the first 15-bytes (QL) or 9-byte (Spectrum) of the very first packet. If the first byte received is not 0xFF on the QL or the expected 0-3 'file-type' byte on the Spectrum, the command will fail with an error. The remaining bytes of the header contain the all-important file-length, and other meta-data – especially important on the QL when attempting to load/execute a Job-type file over the network. *Some older versions of TK2 seem to also expect `LOAD` to operate with a serial header, but are not always sent by the corresponding `SAVE`! Upgrade to a later version of TK2.*
11. **Nothing works after loading TK2 from disk (QL)** – due to the sensitive software timing-loops used by the NET driver, running TK2 from RAM on a basic QL will fail to allow successful communication because RAM access-contention introduces inconsistent delays running software or reading/writing to RAM. *TK2 running in (inherently uncontended) ROM, or running in RAM on a (Super)GoldCard-equipped QL avoids this issue.*

Having now experimented with networking over several years and across diverse QLs, Spectrums and compatible machines, the author rarely experiences problems using the network today – except those created oneself!

1.7. The TK2 enhancements to the QL's NET Driver

Whilst the QL's basic NET device offers some very useful capabilities, almost all of these are enhanced or otherwise augmented with more advanced facilities once TK2 is available. To be frank, anything more than trivial use of the QL requires TK2 to get the best from the machine and this is certainly true in the case of networking.

TK2 replaces the entire QDOS NET driver once installed, maintaining the QLAN protocol specification to allow basic communication even with QLs *without* TK2; it improves the reliability of some features (e.g. network-broadcast) and adds the all-important **FSERVE** server Job with its corresponding **Nx** client-side device-driver.

As mentioned, TK2 requires uncontended RAM or ROM in which to run to avoid the QL's inherent RAM access-contention that disturbs the delicate timing of the driver code.

The basic NET functionality remains as described before, but once the **FSERVE** Job is invoked on one (or more) of the connected stations, it becomes possible to access the file and device resources on the server station as if they were installed locally, through the use of the **Nx** pseudo-device on any TK2-enabled QL client stations.

FSERVE is flexible enough to make almost all the features of the remote device available to the client station (which needs to have TK2 installed, but needn't be running FSERVE itself.) Furthermore, any device-driver linked-in to the serving station can be accessed from the client - not just the obvious file-system devices such as MDV, FLP or WIN.

Thus, SCR, CON, SER, PAR and other devices installed on the serving station can all be accessed from the client using the same general form Nx_<remote_device_spec>. Some truly clever network programming techniques open-up, especially with the likes of the MEM memory-device (thanks to Simon N Goodwin's DIYTK driver.)

Still, most of the value of this client-server capability is in practice found when loading or saving files on the server station. Here are some examples (where Nx means N<server_station-ID>) and note that *nothing else needs to be typed at the serving-station once FSERVE is invoked*:

a) Backup/archiving of entire MDV cartridges across the network to a folder on the server's WIN device, *with a single command at the client station*:
```
WCOPY 'mdv1_' TO 'Nx_win1_backup_'
```
b) Sending print-files to a quality printer attached to the server's SER port:
```
COPY 'mdv1_print_txt' TO 'Nx_ser1hr'
```
– better still, use TK2's SPOOL command in place of COPY.
c) Sending messages to the screen-display of the user sitting at the serving station:
```
OPEN #3,'Nx_scr_100x20a206x118': PRINT #3, "Hello!": PAUSE: CLOSE #3
```
d) Loading EXECutable programs directly from the server's file-system:
```
EX 'Nx_win1_APPS_QED_qed';"mdv1_text_file"
```
e) Directly manipulating the memory (e.g. video-memory) of the server, using MEM:
```
OPEN #3,'Nx_mem': PUT #3\131072: PRINT #3,
FILL$(CHR$(170)&CHR$(0),32767);: CLOSE #3
```
f) Syncing the local real-time clock *(taken from the Minerva docs – who needs NTP?):*
```
f$="Nx_ram1_date_tmp": d%=FOP_NEW(f$)
IF d%>0 THEN CLOSE #d%: SDATE FUPDT(\f$): DELETE f$
```

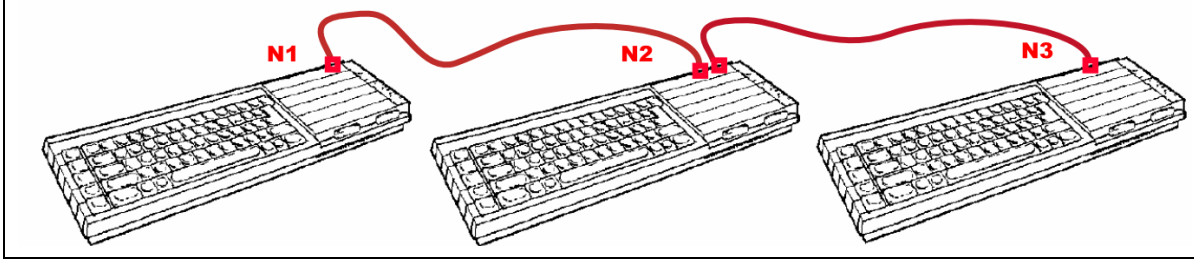
Some of these examples can *almost* be achieved with the basic NET driver, but given that FSERVE runs autonomously on the server station, only the client-side commands need be entered under TK2, as compared to having to enter reciprocal commands on *both* peers without it.

1.7.1. FSERVE's Hidden Gem…

There is a hidden – or at least, not widely documented – feature available when running `FSERVE` that at first inspection may not seem especially useful, but as we'll see shortly really comes into its own.

It turns-out that `FSERVE` running on an *intermediate* station (N2) will happily open on the client's behalf a network-type device targeted at a third *target* station (N3), such as the example picture below.

Multi-hop NET connections using FSERVE



From N1, we can load a file `mdv1_display_scr` on **N3**, *via* **N2**, thus:

```
LBYTES 'N2_N3_mdv1_file', 131072
```

In fact, as far as station N1 is concerned, the file resides on N2. Meanwhile, N2 opens *its own* connection with N3, and then connects back to N1 with the contents of the file actually saved on N3.

Try it - it's slow, to say the least, and of limited value beyond 'intellectual curiosity.'

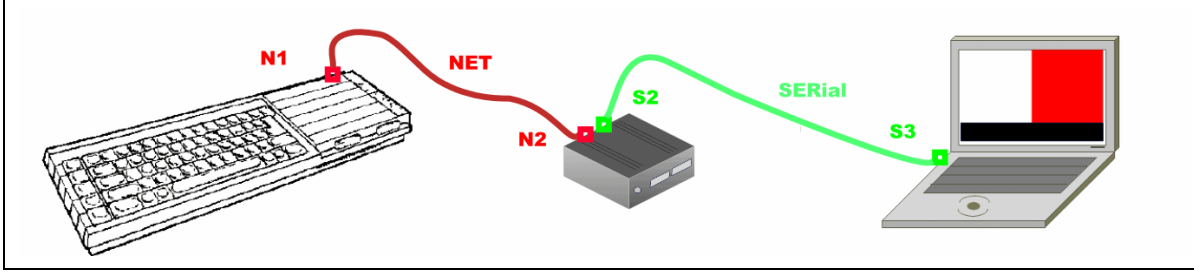
1.7.1.1. Connecting to stations with SERNET…

However, as some QL users have discovered, this same feature can be put to good use when linking stations that don't natively support the Sinclair Network directly, but have either SERial or Ethernet ports available - and the required drivers (**SERNET** or **QLAN over Ethernet**, respectively.)

SERNET is available for the Q40/Q60, the Q68 as well as emulators running on a host PC. It can be run on a basic QL as well, but is not as effective as using the native NET port at the QL end.

Taking the SERNET approach, we can leverage this feature of `FSERVE` (and its SERNET equivalent, suitably named `SERNET`) to effectively 'hop' or bridge between QLAN and SERNET capable devices. We can connect an intermediate server that has both QLAN (as **N2**) and SERNET (as **S2**) available - e.g. a Q68 – and a target station that has the SERNET driver loaded (as **S3**) - e.g. a PC running a QL emulator, as per the following diagram:

Multi-hop NET connections using FSERVE and SERNET



We can then issue the following command on the QL station N1:

```
COPY 'mdv1_file' TO 'N2_S3_win1_file'
```

This will result in `mdv1_file` on the QL being copied to the emulator's `WIN1_` device, even though the emulated QL is only *indirectly* connected to the client station, and itself only has a SERial port and no native QLAN link. This overcomes the native limitation of QL emulators not being able to connect directly to the QL network. *Use of the QLUB Adapter overcomes this limitation.*

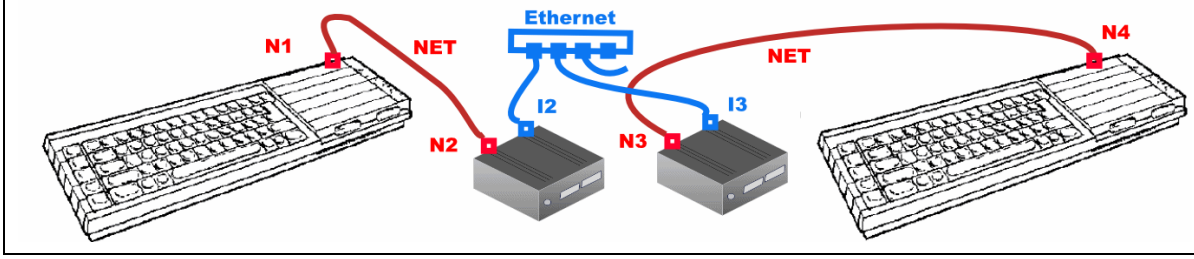
Perhaps more usefully, from the emulated QL **S3**, and with FSERVE running on QL **N1**, we can access the QL's microdrives directly, thus:

```
WCOPY 'S2_N1_mdv_' TO 'win1_backup_'
```

1.7.1.2. Connecting to stations with QLAN over Ethernet…

This feature ***really*** comes into its own when coupled with **a pair of Q68's** with their Ethernet capability running Martin Head's **QLAN over Ethernet** driver, alongside the native QLAN. It is quite possible and sometimes convenient to house your QL network stations in different parts of the building, using IP to traverse the distance between (e.g. via WiFi or PowerLine) such as the example below:

Multi-hop NET connections using FSERVE and QLAN over Ethernet



For example, the author has a number of QLs and Q68s both in the office upstairs and in the shed at the end of the garden, inter-connected via Ethernet PowerLine adapters into which a Q68 is connected at each end via its Ethernet port. Each Q68 is in-turn connected to separate QLAN network 'segments' (as `N2/I2` and `N3/I3`) with several QLs on each segment and, once both `FSERVE` ***and*** `IFSERVE` are running on the Q68s, and `FSERVE` at the target QL (`N4`), I can access the target station from a QL `N1` with:

```
LBYTES 'N2_I3_N4_win1_display_scr',131072
```

The effective transfer rate is little different than if there were no Ethernet 'hop' at all – just a dual QLAN route - with the Ethernet hop appearing to add almost negligible additional latency.

Given the native **routing** capability of TCP/IP, the possibility opens-up of hosting FSERVE services over the public WAN – a topic for a future article…

1.8. Interconnecting the QL and the ZX Spectrum

Notwithstanding the reported challenges with communication between the QL and Spectrum over the Sinclair Network, there are some useful activities that become possible when these two machines are linked that are well-worth persevering-with given that the (basic) QL and Spectrum/Interface-1 share an entirely compatible protocol for their respective network implementations.

How, then, is it that most users were unable to make the Spectrum/QL connection work for them, even though Spectrum to Spectrum network comms seemed relatively usable?

During the author's investigation several bugs were identified in the Interface-1 network code, the most significant of which was the observation of inconsistent timing when *sending* certain bit-patterns *from* the Spectrum. The cause was found to be 'IO Contention' when writing the next transmission-bit to the Interface-1 ULA register.

This finding aligns with the general consensus that QL *to* Spectrum works reasonably well, but you could rarely receive anything back *from* the Spectrum, severely limiting its usefulness.

This behaviour has since been diagnosed and replacement Z80 routines developed and tested to avoid the ULA Contention issue, among other subtle faults.

The ULA of the Interface-1 actually provides **hardware assistance** in the detection of the 'START' bit, which conversely, is done entirely **in software** on the QL. This allows for less 'jitter' in the START-bit detection on the Spectrum (perhaps just 1 or 2 Z80 WAIT states – less than 570ns) compared to the QL (between 0 and 4us late) and is likely why *receiving* is less problematic for the Spectrum than for the QL.

Replacing the Interface-1 ROM is not trivial - to put it mildly – due, in part, to the exceptionally tight-spacing inside the elegant Interface-1 housing. None-the-less, the author has observed reliable comms between these machines with the modified Interface-1 ROM routines in-place and has shared the modified Shadow ROM image online for any users willing to hack their precious Interface-1 hardware.

Alternatively, using slightly revised 'Timing Constants' in the TK2 NET driver, the QL can be made to better accommodate the Spectrum's wobbly bit-timings without recourse to hacking the Interface-1. The result is not 100% perfect, but a good step forward.

Fortunately, the QL fitted with a Super-GoldCard (and possibly, the GoldCard) as well as the QXL can already cope with the Spectrum's send-timing anomaly without further adjustment, though again, not always 100% reliably. The Q68 running the ND-Q68 driver, and now, the ND-Nxt driver on the Next/QL-Core on the other-hand, seem to take the Spectrum's wobbles in their stride…

However it is achieved, once a reliable connection between QL and Spectrum is established, you can:

a) Backup your ZX microdrive cartridges to reliable QL storage
b) 'Boot-strap' the Spectrum with a Launcher for game-files hosted on the QL
c) Develop Spectrum software more conveniently within an emulator on the QL (e.g. Speculator or the brilliant ZM/x) and transfer back to a 'real' Spectrum
d) Transfer screen-dumps of your favourite Spectrum games and render them on the QL (with some additional QL software)
e) Explore multi-player/multi-*platform* network games

There are a number of differences that need to be considered when transferring files between these two diverse platforms, some of which are discussed below, before we close this first article on Spectrum/QL networking:

1. Spectrum/Interface-1 vs QL **file-headers**
2. 'Foreign' programs and Spectrum **BASIC tokenisation**
3. Screen display **file-formats**

1.8.1. File-headers

The first challenge to overcome when transferring files between the two platforms is taking-account of differences between their respective 'file-headers.' Such headers are typically only present on files that need them, as opposed to simple 'serial-access' data-files which do not – much like the concepts of 'header' and 'header-less' tape-files, familiar to Spectrum users.

On the QL, there are two forms of file-header, the usual 'directory-device' type headers at 64-bytes each, and the simplified 'serial-file-headers', at 15-bytes (including an initial flag-byte, 0xFF) which include a subset of the 64-byte variety. As the QL's NET device is of the 'serial' type, the shorter file-header is used.

On an unexpanded Spectrum, the tape-system uses 17-byte file-headers, which include the 10-byte/characters of the filename. The Interface-1 introduced a smaller and slightly modified 9-byte file-header used on microdrives and the network, which drops the filename (which on microdrives, is instead stored in the 'record-header' and not sent at all over the network.)

It proves easier to manage the differences at the QL end, rather than at the Spectrum and, fortunately, the QL can accommodate 'header-less' files easily with the `COPY_N` procedure when needed.

Receiving files from the Spectrum is straightforward, as the QL will treat the incoming file as header-less by default when `COPY` is used with a 'serial' device such as NET – only requiring a file-header if `LOAD`/`LBYTES` or `EXEC` (don't!) is used to load the Spectrum file.

Thus, where `Z` and `Q` are the station-IDs of the Spectrum and QL respectively, the following commands will correctly transfer `<filename>` to the QL, storing the Spectrum file, along with its 9-byte header on the QL as if it was part of the file image; the QL sees a file-length 9-bytes larger than the Spectrum records in its header:

Spectrum: `MOVE "m";1;"<filename>" TO "n";Q` *or*
Spectrum: `SAVE "n";Q SCREEN$` *(or whatever type of file this represents.)*

QL: `COPY 'neti_z' TO 'win1_ZXFiles_<filename>'`

To send the file back again, we must persuade `COPY` *not* to add a QL serial-file-header, so we use the `COPY_N` variant, thus:

QL: `COPY_N 'win1_ZXFiles_<filename>' TO 'neto_z'`

***Spectrum**: `MOVE "n";Q TO "m";1;"<filename>"` *or, better-still*
Spectrum: `LOAD "n";Q SCREEN$` *(or whatever…)*

** The Spectrum `MOVE` command was deliberately 'crippled' in the Interface-1 ROM to limit reading or writing 'non-PRINT' type files from/to microdrive to discourage software piracy. Thus, without a revised ROM, making use of the `*MOVE` extension-command, or else direct manipulation of the Interface-1 microdrive channel-block to force a non-PRINT file-type whilst writing, the saved file will be accessible only through the `OPEN#` command – `LOAD` would fail with 'Wrong file type.'*

Here, any *QL* file-header embedded in the file `win1_ZXFiles_<filename>` is stripped before transmission; it would only confuse the Spectrum. As long as the copied file still includes the Spectrum file-header as the first 9-bytes, it will be recognised correctly by `LOAD` when received on the Spectrum.

In addition to these simple S/Basic commands, you might consider the tools included with the **Speculator** emulator distribution, or else the **QSPEC2** utility suite, by Dave Barker – you may need to hunt them down on the Internet.

1.8.2. Foreign programs and Spectrum BASIC

The QL can't of course run Spectrum (Z80) machine-code programs directly, nor load Spectrum BASIC program-files directly into the QL's S/Basic interpreter. However, with a suitable Spectrum emulator such as the highly flexible **Speculator** (and its extensive toolkit) by Dave Walker and Simon N Goodwin, or the truly impressive **ZM/x** series of emulators (esp. **ZeXceL**) by Davide Santachiara & Marco Ternelli of Ergon Development, any program-files transferred to the QL from the Spectrum as described above can, with some additional steps, be loaded in to the emulator, tested and run.

Most of the Spectrum programs you might want to load in the emulator are probably available for download from the various online Spectrum repositories (e.g. **World of Spectrum**: https://www.worldofspectrum.org/). However, you may have your own code and BASIC programs

stored (precariously!) on Spectrum microdrives that you wish to develop further, 'port-over' to QDOS or otherwise enjoy on your QL.

I developed myself several, fairly substantial Spectrum BASIC programs in my youth (when I should have been revising for my O-Levels!) and would be loathed to have to re-write them from scratch in order to continue their development on my QL.

There are a few utilities available to convert the Spectrum's tokenised BASIC files into plain ASCII and thus ready for porting to S/Basic, but most of the titles the author finds online are PC-based (`ZmakeBAS.exe` is a very good example, if a bit fiddly.)

However, writing a BASIC de-tokeniser on the QL to do this work would be a relatively straightforward task (using the ZmakeBAS source files as a starting point) – taking care in the process of conversion to S/Basic variants of similar commands. Alternatively, the QSPEC2 utility suite mentioned previously includes a BASIC de-tokeniser/lister command called `SLIST`.

One last note – the Spectrum uses the 'CR' code `CHR$(13)` as its line-terminator rather than the QL's Unix-like approach of using 'LF' `CHR$(10)` and most of the Spectrum word-processor applications seem to stick with CR (or else use fixed line-lengths). Thus, if you plan on transferring *documents* created on the Spectrum, you'll then need to patch them to use the QL's line-terminator to make them legible within a native QL editor.

1.8.3. Screen Display format

The display-handling between Spectrum and QL is significantly different, but there is some fun to be had in converting the Spectrum's pixel/attribute-based 6912-byte display-file format to the linear-progressive, pixel-based QL 32KB equivalent (best in `MODE 8`, or else the Thor's `MODE 12`) – they make wonderful desktop wallpapers for a QL equipped with the Extended/Pointer Environment and many are available for download from sites such as World Of Spectrum.

There are several display conversion tools available already, including, again QSPEC2's `SVIEW` command or else one can be readily coded in QL SuperBASIC.

For example, I use a QL-rendered screen-dump of Ultimate's iconic "JetPac" as my test-file for exercising the network – copying the display back and forth - whilst developing new versions of the network drivers; it brightens-up an otherwise tedious and repetitive process!

This brings us to the end of the first article.

1.9. Postscript

In the second article we shall take a closer look at the standard NET protocol and the TK2 extensions to it, this time from the perspective of each OS – QDOS and the Shadow ROM, as well as the network hardware itself as implemented in each machine.

In addition, the **QLUB Adapter** will be presented, with the simple schematic for connecting a QL emulator to the Sinclair Network via the host PC's USB port using the QLUB's in-built AVR microcontroller.

For users of the **Q68**, the simple circuit needed to connect the Q68 to the Sinclair Network will be presented also.

In the third article, the potential future of networking the QL, Spectrum and their compatibles will be presented – and how the Sinclair Network Standard might be further ported to other, *non*-Sinclair machines of the era via some additional hardware.

1.10. Appendix

In this appendix, the new Network driver for the **ZX Spectrum Next**/**QL-Core** (ND-Nxt) is presented along with the **Joy2NET adapter** design to assist any lucky Next owners get connected to the Sinclair Network.

1.10.1. ZX Spectrum Next/QL-Core Network Support

In late 2024, a brand new FPGA core for the ZX Spectrum Next was released by Theodoulos Liontakis allowing both KS1 and KS2 Next users to enjoy the world of the QL.

The **QL-Core** is specifically configured to enable use of one of the Next's multi-purpose Joystick ports in its 'bit-banging' mode so as to support the Sinclair Network. This is just one of the many features the QL-Core offers Next users – more details can be found on the official SpecNext site: https://www.specnext.com/sinclair_ql-qs/

The following text is copied from the original announcement and provides a short description of the network adapter/driver solution, as demonstrated at the CRASH Live! event, November 2024:

The Sinclair Network-Driver software (**ND-Nxt**) plus an adapter (**Joy2NET**) allow connection of your Next to both the QL as well as the ZX Spectrum/Interface-1 at around 90 kbps - without the need to purchase and attach an Interface-1 to the Next itself.

Connection to a Sinclair Network allows for simple transfer of files between legacy Sinclair machines and the Next, as well as the possibility of new network-based multi-player games!

Connection is made via cheap and robust 2-core cabling, with simple 3.5mm mono self-terminating jack-sockets - and with none of the cost and complexity of the cabling typically required by serial-port and other network technologies of the era.

Tony Tebby's Toolkit-2 File-server/client is fully supported between the Next's QL-Core and other QLs, further simplifying file-transfer and other advanced Net-RPC type functions.

The ND-Nxt driver is a brand-new port of the ND-Q68 driver previously developed for the excellent Q68 FPGA-based QL - both drivers include 80%+ of Tony Tebby's Network code and the efficacy and simplicity of this low-cost network solution is a tribute to TT and Sinclair Research Ltd's original design!

The development of a *native* ZXNextOS network driver is also planned to allow the Next's default 'personality' to access the Sinclair Network, further enhancing connectivity options for Next users.

The simple adapter required to connect the Next to the Sinclair Network can be constructed using readily available components for less than £10 for any users wishing to build it themselves. Alternatively, a professionally assembled adapter is planned to be made available for purchase in due course for those users less inclined to build one themselves.

The Joy2NET adapter schematic and bill-of-materials appears below, whilst the ND-Nxt driver software has been included within the ready-to-run demo **QXL.WIN** disk image available from the SpecNext/QL-Core site; *the source-code will be made freely available for anyone to further adapt and develop once it is decided where best to host it.* The driver will continue to be maintained and new versions made available via the SpecNext site.

The driver can be loaded from within the BOOT file by adding a simple command, such as:

```
LRESPR "win1_<path_to_driver>_ND-Nxt_dvr_v<version>_bin"
```

Where **<version>** is "**110**" as of January 2025.

1.10.2. Joy2NET Adapter

The adapter needed to connect the Next's Joystick Port #1 to the Sinclair Network requires a handful of components that can first be assembled on prototyping breadboard or else committed to PCB. The voltage levels are all 5V, low current (c 25mA) and relatively low frequencies (in modern computing terms.)

The main purpose of the adapter is to condition the voltage levels and termination resistance suitable for the network and to match the voltage/logic levels expected by the Joystick port. It also provides a **small switch** to manually disable the adapter during the Next startup, prior to starting the QL-Core and loading the ND-Nxt driver software - at which point the adapter can then be enabled via the switch.

No harm will come from leaving the adapter permanently enabled, but if a network cable is also left attached, strange results may be observed on both the network (corrupted packets) and in the Next UI (triggering of the secondary 'fire' button) until the QL-Core is started and the driver loaded.

The design presented here has been tested extensively with both the KS2 and KS1 (N-Go) Next variants successfully. Several different physical designs have been assembled and work well – photos appear below of one such example, but users may wish to experiment with alternative cable lengths and optional components to suit their own setup and preferences.

Example Joy2NET Adapter Design

The Joy2NET adapter can be used alongside standard/passive Atari-style joysticks in port #1 if a pass-through cable/socket is added to the design (as shown). A standard Atari joystick is also supported in port #2 in any case.

However, the more sophisticated Sega Mega Controller (MD) is **not** supported *in either port* simultaneously with the network as it uses some of the same pins of the 9-pin D-type Joystick socket as the network circuitry. In any case, the QL-Core itself does not support the MD controller type as no QL software has been developed to-date to read the additional MD controller buttons.

If you wish to attach an MD type joystick to the pass-through connector of the adapter, you can leave the adapter attached with the switch set to 'Joystick' mode and then use the Next's other personalities as needed without disconnecting the network cables (obviously, no network access will be available whilst in joystick mode.)

1.10.3. Joy2NET Schematic and BOM

A fully-specified adapter will require the following parts. Some parts can be dropped if less functionality is required – noted with an asterisk *

Bill of Materials

Component type	Qty	Value	Comment
Resistors	1x	47 Ω	
	2x	330 Ω *	Only **one** 330 Ω required if only one net connection is needed
	1x	1k Ω	
	1x	3k9 Ω	
	1x	10k Ω	
Capacitors	2x	1n F *	Only required when running the net at *higher bit-rates*
Transistors	1x	BC546 NPN	Or equivalent (e.g. BC337)
	1x	BC556 PNP	Or equivalent (e.g. BC327)
Connectors	1x	DE-9 female plug	E.g. cut-off from 9-pin Joystick Extension cable
	1x	DE-9 male socket *	Needed only for joystick pass-through
	2x	3.5mm mono-**switched** Jack socket *	Only one **non-switched** mono Jack-socket required if only one net connection is needed
Other	1x	Small prototyping box	E.g. Hammond project box, 1551MBK
	1x	Small PCB/prototyping board	Shape to fit inside the chosen prototyping box
	5cm	9-core cable	E.g. cut-off from 9-pin Joystick Extension cable
	10cm	9-core cable *	Extra length needed only for joystick pass-through
	1x	DPDT (micro) switch	

Schematic

The following schematic includes some of the components internal to the KS2 Next itself, solely for reference purposes – only the items in grey/black actually form part of the Joy2NET adapter.