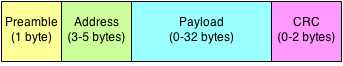
Some ideas:

[Nrf24 sniffer:](http://yveaux.blogspot.com/2014/07/nrf24l01-sniffer-part-1.html)

**Regular Package**



Preamble 🡪 used to identify incoming packets.

CRC 🡪 checks the integrity of the message after reception. The CRC is calculated over the whole message, excluding the preamble and the CRC itself.

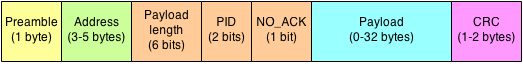
The radio must be told upfront what the length of the payload will be, as a regular packet contains no indication of the length.

When a message received matches the receiver address of the radio and the CRC is found valid or disabled, the radio will store the payload in an internal FIFO for reception by the host (microcontroller or whatever connected to the nRF24 using SPI).

**Enhanced Shockburst**

'Enhanced Shockburst' advantages:

* Payload lengths can be set dynamically and are part of the message
* The receiving node can automatically send an acknowledge to the sender to indicate that the message has been received correctly. The sender will automatically retry transmission a number of times when the acknowledge is not received within a configurable timeout (Nordic calls this "Automatic packet transaction handling")



Packet identifier (PID) 🡪 detect retransmissions and a flag to suppress sending acknowledge packets on a per-packet basis.

The CRC is no longer optional. As with the regular packets, a message received must match the receiver address of the radio and the CRC must be valid to have the payload stored in the internal FIFO, otherwise the message will be discarded.

**Promiscuous reception**

* When building a wireless network of many nodes you choose a single RF channel which all nodes will operate on.
* Furthermore each node shall have a unique address within the network (unless you're only broadcasting).
* Each node can have an arbitrary address, but normally you choose a base address which is identical for all nodes, and a node address which is unique for each node.

Should the nRF24 support promiscuous addressing, then the base address would have to be configured in the chip and it should be instructed that the node addressing consists of 1 byte.

The nRF24 would then start listening for messages. All messages with matching base address and valid CRC would then be stored

But the nRF24 does not support promiscuous listening. With some trickery we can however instruct it to capture all messages for a certain base address, including the ones with invalid CRC's !

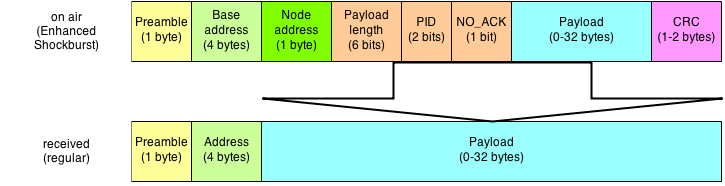
The following configuration of the nRF24 sniffing radio is required:

* Set the nRF24 receiver address to just the base address
* Disable Enhanced Shockburst
* Disable CRC checking
* Configure a fixed payload size

The first 3 items will cause the nRF24 to capture anything that matches the base address of the network we're sniffing. As Enhanced Shockburst is disabled, the radio will not determine the payload size from the message anymore and we just have to store a fixed amount of data starting from the Node address. We just fake the message received is a regular packet, while in fact it is an Enhanced Shockburst packet!

CRC checking has to be disabled of course, as the nRF24 doesn't know the payload length anymore and CRC calculation would fail, rejecting all messages.

The process is illustrated by the image below (again, taking the 4+1 addressing scheme as an example):



The payload received will start with the node address the packet was transmitted to, so effectively we've created a promiscuous listening nRF24!

As the nRF24 will no longer dissect the packet for us, we'll have to do it ourselves (actually let Wireshark do it for us).

The original length of the payload should be extracted (the 6 bit payload length), the full address recreated and CRC calculated over the data received. When the CRC is invalid the message can be marked, which gives us a nice indication of the link quality.

As the nRF24 header is not a multiple of 8 bits, all payload & CRC bytes will be bit-shifted directly adjacent to the packet control field. This is inconvenient when inspecting the raw bytes in a packet, but Wireshark will solve this nicely for us.

Limitations

There are however some limitations to this method which you should be aware of:

* The maximum length of a payload is limited by the nRF24 to 32 bytes. As we 'shift' the reception of the payload towards the start of the packet and also want to include the CRC in the payload, the effective size of payloads captured will go down by a number of bytes. The amount depends on the number of bytes in the node address, the fixed packet control field and the length of the CRC.
* When fixed payload size of the sniffer is set to a value (much) larger than the actual payload size in the packet, then subsequent packets might be missed. When the nRF24 is still capturing the fixed size payload and a new packet arrives, this will not be detected. A good example for this are the auto-acknowledge packets which can be enabled in Enhanced Shockburst mode. After the target node receives the packet (and it will know the real size of the payload a lot faster than the sniffer will) it switches its radio from receiving to transmitting mode in only 130us. Then it sends out the acknowledge packet. Reception of 32 bytes at 1Mb/s takes 32\*8us = 256us, so acknowledge of a short message will definately be missed. For acknowledge packets this isn't much of a problem as the transmitter will retry to send the identical message when no acknowledge was received and we will know it missed the acknowldge.
* This method works well for Enhanced Shockburst packages, as they have the payload size embedded in the message. For regular messages the length will have to be known or can be determined by testing different lengths until the CRC matches. With CRC disabled the payload size can only be guessed.
* The addressing scheme must use a base address in the high address byte(s), and node address in the low byte(s), as explained

[**Software**](http://yveaux.blogspot.com/2014/07/nrf24l01-sniffer-part-2.html)

The required software is split in the following parts:

* Sketch running on the Arduino: configures the nRF24 radio and reads any messages received using SPI.

It determines the actual payload size for each message and adds a microsecond-resolution timestamp.

The messages are buffered in the Atmega328 and sent to the host over a serial connection. It keeps track of the number of lost messages, when the buffer is full.

* Console application 'Nrf24Sniff.exe' (currently only Windows) which sends configuration data to the Arduino sketch and reads captured packets from the sketch, using a serial connection.

The packets received are forwarded in libpcap-format to a named pipe. Wireshark will be instructed to capture from this named pipe (too bad it cannot capture directly from a serial connection).

* Wireshark to visualize the captured packets, filter them and gather statistics.
* Wireshark will be extended with one or more plugins (called dissectors, currently only precompiled for Windows) which recognize the nRF24 packet format and take it apart. Protocols which use the nRF24 as transport layer (e.g. MySensors) require a separate dissector.

**Running**

1. Connect the hardware running the sniffer sketch to your PC (either using an RS232 connection or a serial-to-USB bridge as present on the Arduino Uno).

N.B: Note the comport assigned by Windows (the same port as in the Arduino IDE). Assure the Arduino IDE's Serial Monitor is not running as it will block access to the serial port.

1. Nrf24sniff.exe can be started from a cmd-window. It supports a number of commandline parameters which control its behaviour (display them using 'Nrf24Sniff.exe -h'):

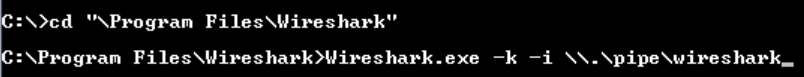
For example, when monitoring MySensors 1.4beta traffic (at non-default 1Mb/s), using the Arduino Uno sniffer connected to com17, I run the tool as follows:

Nrf24Sniff.exe -P17 -c76 -r0 -l5 -p4 -a0xA8A8E1FC00 -C2

The address length is set to 5 bytes (-l5), of which 4 bytes are used as base address (-p4) (Please refer to part 1 of this series for a description of addressing schemes). Passing only the comport would have been sufficient in this case, as the rest of the parameters are all set to defaults.

Now run it:

1. Keep it running and open another console window to start Wireshark:



This instructs Wireshark to start capturing from interface (-i) \\.\pipe\wireshark (that's the named pipe Nrf24Sniff.exe will be writing to) and to start capturing immediately (-k). More commandline options for Wireshark can be found here.

After Wireshark starts capturing, the Nrf24Sniff console window will display a 'Wait for sniffer to restart' message for a few seconds (at most):

N.B: If it hangs here, restart the sniffer hardware manually by pressing the reset button on the Arduino (I'm working on that...).

1. As Wireshark has no clue how to interpret the data coming through our named pipe, we have to tell it which dissector to use.

Switch over to Wireshark and select Edit -> Preferences.

In the preferences window, open the protocols tree, scroll down and select DLT\_USER.

Press the edit button for the 'Encapsulations table' and press New. Enter the data as in the screenshot below:

Press OK, OK, OK. Any packets on air that match the radio configuration should now be captured in Wireshark!

The Nrf24Sniff console shows a count of packets captured and lost:

1. http://yveaux.blogspot.com/2014/07/nrf24l01-sniffer-part-3.html