Olso net Communications

SA:

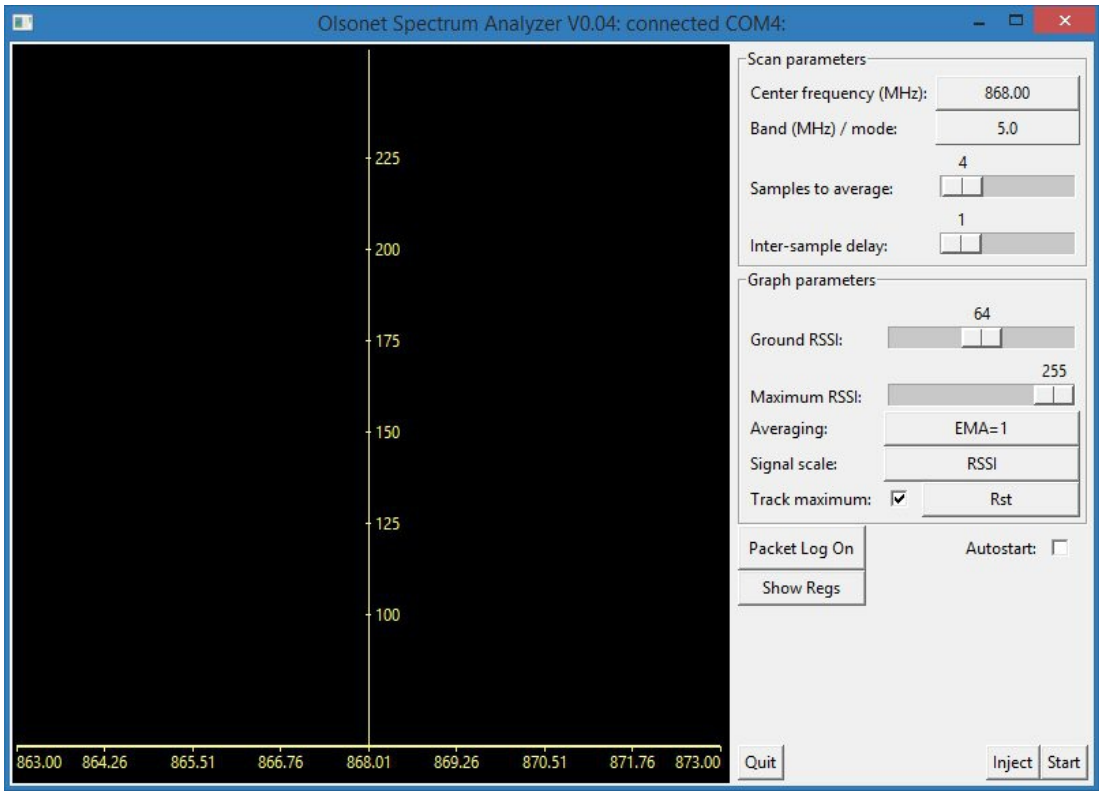
a Spectrum Analyzer/Sniffer for PicOS Networks Based on CC1100 RF Modules

version 0.84

January 2015

1 Introduction

SA is a simple sniffer/RF-scanner based on standard CC1100 hardware, i.e., the underlying praxis runs on a standard node equipped with a CC1100 module, e.g., the old “Warsaw” box or the new CC430-based node. Needless to say, SA is rather far from an industrial grade spectrum analyzer, but it does offer a few features that may prove useful in our development work. Its OSS program, having been written in Tcl/Tk, is open for extensions that will materialize quickly, as soon as we find them useful.


Olsonet Spectrum Analyzer V0.04: connected COM4:

Scan parameters
Center frequency (MHz): 868.00
Band (MHz) / mode: 5.0
Samples to average: 4
Inter-sample delay: 1

Graph parameters
Ground RSSI: 64
Maximum RSSI: 255
Averaging: EMA=1
Signal scale: RSSI
Track maximum: ☑ Rst

Autostart: ☐

Figure 1: The main window as it shows up immediately after startup.

As of now, the program offers:

1. Flexible hardware settings of the CC1100 options (registers), which can be easily edited and saved in a database for future use. A handy collection of hover tips makes it easy to modify the register contents without having to consult the documentation.
2. Two graphical scan modes: a) *band scan*, whereby a range of frequencies is periodically swept for RSSI readings, and b) *single frequency scan*, whereby a single frequency is monitored for RSSI and (optionally) for packets.
3. A packet reception option available as a special mode of the single frequency scan, where the program can receive packets and report them in a special window (optionally logging them to a file).
4. A packet injection option (available as part of the single frequency scan), where you can predefine collections of packets to be injected into the network manually or automatically at prescribed intervals. Selected fields of those packets can be incremented/decremented with each injection. Similar to register configurations, editable packet layouts can be stored in a database for future reference.

2 Prerequisites

You need a "Warsaw" node. Compile the SA praxis (the board is WARSAW_SA) and flash it into the node. Connect the node to your PC via the standard FTDI (UART-USB) dongle.

The OSS program for the praxis is a Tcl/Tk script. The simplest way to make it easily and independently executable is to package it with *freewrap*. (see the *mkexe.sh* script in the praxis directory). The script needs Tcl/Tk 8.5, but it doesn't depend on any exotic features (that wouldn't be found in *freewrap*). Needless to say, you can also execute the script directly using your existing installation of Tcl/Tk (again, it must be at least 8.5).

Run the script (or the pre-wrapped executable). After a (hopefully not too long) while it should *automatically* locate the node and connect to it. There is no connect button, there is no need to look up COM ports, or anything like that! The connection status is indicated in the main window's title. If it says "connected", it means that the script is talking to the node. The main window (in its virgin startup shape) is shown in Figure 1.

> **Note:** you can force the program to connect to a specific port by putting the port number as a call argument. This is only possible if the program is called from a command line. Generally, you can put any number of ports into the arguments, and then only those ports will be tried. For Windows, those are COM port numbers; for UNIX, they are interpreted as the numbers of */dev/ttyUSB* devices. You can also use full port/device names, e.g., *COM5:* or */dev/ttyUSB1.*

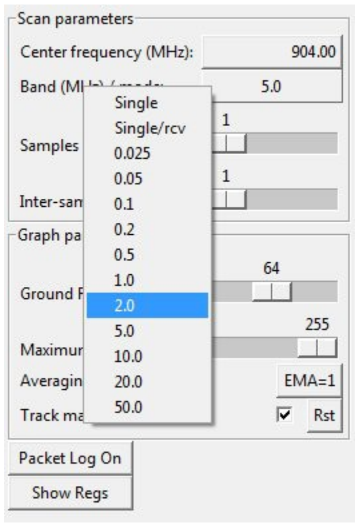


Figure 2: Mode selection.

When the program starts for the first time (and it has no *rc* file) its parameters take some default values (some of them are shown in Figure 1 as preselected options). In particular, it uses the default configuration of registers (named *Built In*), which you can see when you click on the "Show Regs" button. Their setting closely matches our default setting, which means that you are immediately ready to sniff within our (typical) networks.

For special (dedicated) applications the program may come preconfigured with a specific set of defaults, including different frequency and mode selections as well as different default register settings, e.g., corresponding to different transmission rates, and so on.

The program stores its *rc* file (named *.sapicrc*) in the user's home directory (whatever that means). Under UNIX, and also under Cywgin, when SA is run using Cygwin's "native" Tcl/Tk, the situation is clear. When run from the wrapper, or in any other circumstances when Tcl/Tk does not know about Cygwin, the home directory is typically *C:/Users/your_username*.

3 A band scan

By default (strictly speaking by the "standard" default), the program is set to the *band scan* mode. The center frequency for the scan is set to 904 MHz, and the band is 5 MHz on each side. When you click the "Band/mode" menu (see Figure 2), you will see a list of options. The top two entries select the other (*single-frequency scan*) mode. Don't choose them for now.

With the default settings (without touching any controls), click "Start" (in the right bottom corner). You will see some activity in the window (see Figure 3). The dancing green dots represent the current (momentary) readings of RSSI at the given frequencies; the red curve shows the maximum readings (obtained so far) at the respective locations. The green numbers in the left upper corner tell the maximum reading obtained so far, the RSSI value, and the frequency.

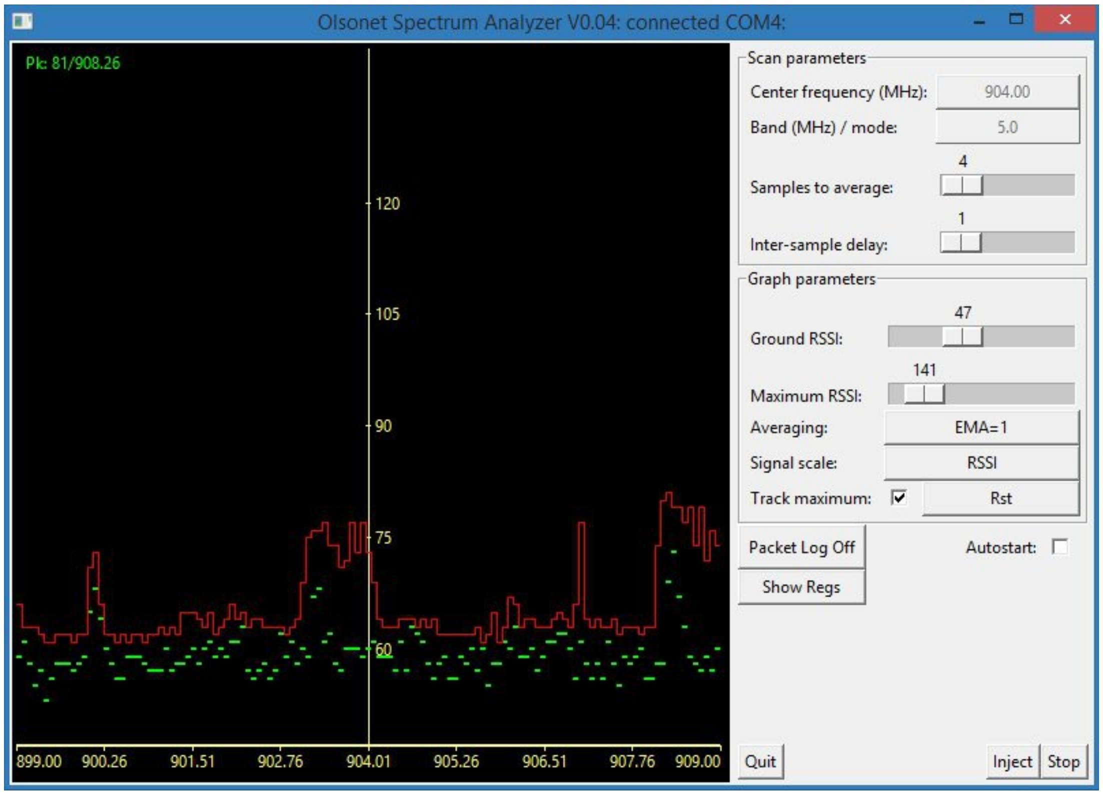


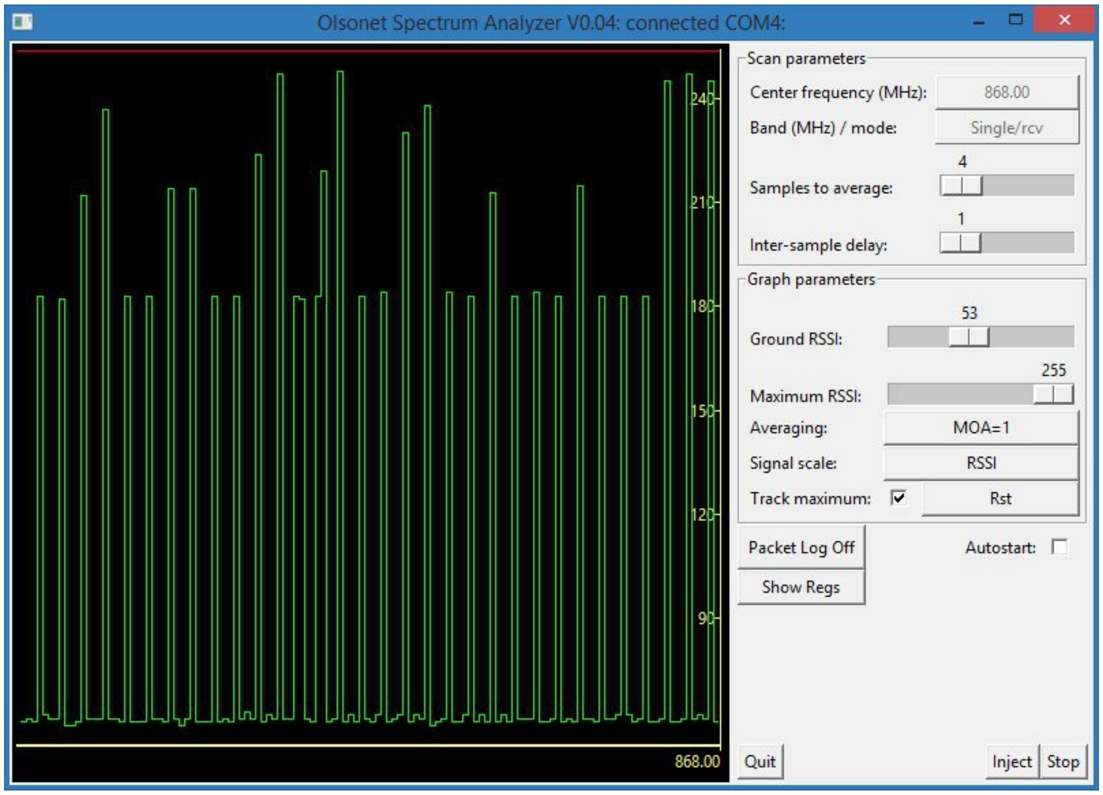
Figure 3: A band scan in progress.

During a scan, some controls (the ones under "Scan parameters") stay frozen, but you can play with "Graph parameters". In particular, you can reset the red (maximum) curve by hitting the "Rst" button (at "Track maximum"), so the maximum will be tracked from scratch (this will also affect the numerical maximum displayed in the left upper corner). You can completely switch off maximum tracking by unchecking the box located to the left of the "Rst" button. Not everything necessarily makes a lot of sense. This is still a bit tentative, and things may change.

Within "Graph parameters", the sliders labeled "Ground RSSI" and "Maximum RSSI" allow you to redefine the vertical scale, i.e., the bounds of the RSSI range. The "Averaging" button opens yet another slider that you can use to dampen the changes of momentary RSSI samples (the dancing green dots). The default (EMA=1) means no damping, i.e., the green dots in fact represent individual RSSI samples arriving from the node. With EMA=4, for example, every dot is an exponential moving average of the last 4 readings. As you increase EMA, you will see that the dots calm down. At the highest setting of 64 they hardly move at all (unless your environment exhibits long-duration RF activities at the scanned frequencies).

By default, the vertical axis is scaled in RSSI (formally from 0 to 255). When you click the button at “Signal scale”, you will switch to dB (subsequent clicks will toggle), which is basically the same thing, except that the values are displayed relative to the current “Ground RSSI”. Please keep in mind that it is all relative (so it doesn't really matter); in particular, I did not try to transform the values into dBm, because they still would say little about the actual signal strength at the antenna. RSSI is more straightforward, because these are the numbers that a PicOS praxis would actually see when receiving packets.

The topmost two menus from “Scan parameters” are not difficult to comprehend. The two sliders, however, do require a bit of explanation. “Samples to average” defines the number of samples (RSSI readings) that will be averaged at the node before being sent to the script as one sample. For example, if you set it to 4, then the node will read 4 times the RSSI at every given frequency and will send you the average of those readings as a single sample. This is slightly different from the other kind of averaging described above (which is done within the script).


Olsonet Spectrum Analyzer V0.04: connected COM4:

Scan parameters
Center frequency (MHz): 868.00
Band (MHz) / mode: Single/rcv
Samples to average: 4
Inter-sample delay: 1

Graph parameters
Ground RSSI: 53
Maximum RSSI: 255
Averaging: MOA=1
Signal scale: RSSI
Track maximum: ☑ Rst

Autostart: ☐

Figure 4: A single-frequency scan.

To appreciate the difference, you have to realize that the node scans the prescribed frequency in sweeps. The samples averaged by the script (EMA) come from different sweeps, so they are separated by relatively long intervals (depending on the inter-sample delay, as explained below). On the other hand, the averaging done by the node refers to temporary close samples, ones having a much higher likelihood of belonging to the same activity.

The “Inter-sample delay” is the delay (in PicOS milliseconds) following every frequency change and preceding the first RSSI sample read for that frequency. The minimum is 1 millisecond.

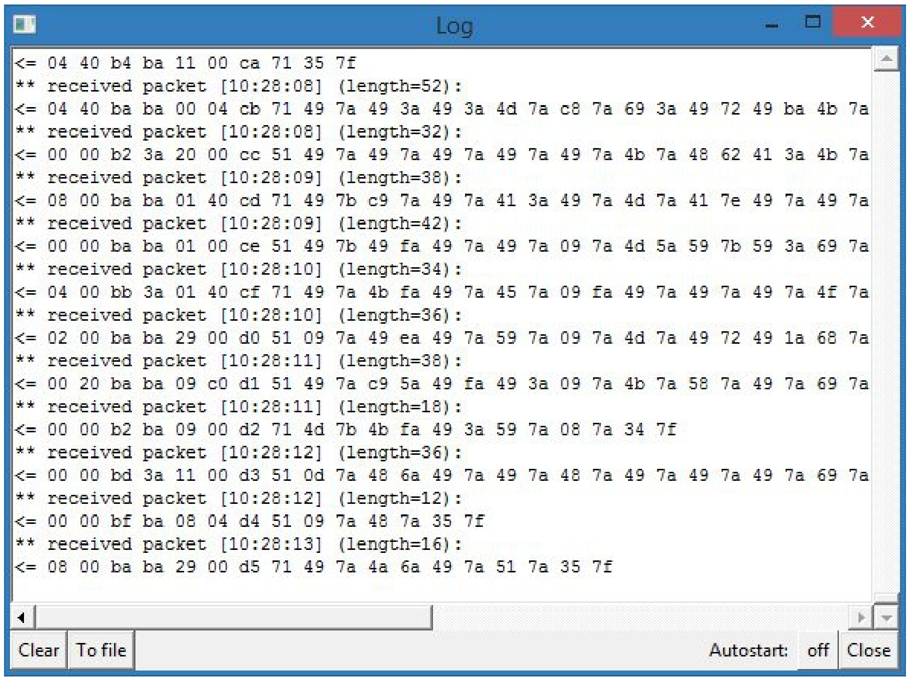
Note that the inter-sample delay does not apply to multiple (averaged) samples collected at the same frequency, which are collected very fast (in a CPU loop). Note: the way it is done at present probably makes little sense, because the RSSI update rate is well below the reading frequency.

The full sweep consists of 129 samples (this is the actual resolution of the x-axis). Thus, at the minimum inter-sample delay, one sweep takes about 0.25s.

4 A single-frequency scan

Click “Stop”. The green dots will stop their dance and freeze. After a short while (some minor skidding occasionally happens), the green LED on the node will stop blinking, the “Stop” button will become “Start” again, and “Scan parameters” will become modifiable. Click the band selection menu and choose “Single/Rcv”. The graph layout will change a bit. Then click “Start” again.

You will see a shifting sequence of RSSI samples for the selected “Center frequency” with the maximum (so far) now represented by a straight horizontal red line (see Figure 4). While the “Ground RSSI” and “Maximum RSSI” settings retain their meaning, “Averaging” now refers to something slightly different (note that the button's title has changed from “EMA” to “MOA”).



```
<= 04 40 b4 ba 11 00 ca 71 35 7f
** received packet [10:28:08] (length=52):
<= 04 40 ba ba 00 04 cb 71 49 7a 49 3a 49 3a 4d 7a c8 7a 69 3a 49 72 49 ba 4b 7a
** received packet [10:28:08] (length=32):
<= 00 00 b2 3a 20 00 cc 51 49 7a 49 7a 49 7a 49 7a 49 7a 4b 7a 48 62 41 3a 4b 7a
** received packet [10:28:09] (length=38):
<= 08 00 ba ba 01 40 cd 71 49 7b c9 7a 49 7a 41 3a 49 7a 4d 7a 41 7e 49 7a 49 7a
** received packet [10:28:09] (length=42):
<= 00 00 ba ba 01 00 ce 51 49 7b 49 fa 49 7a 49 7a 09 7a 4d 5a 59 7b 59 3a 69 7a
** received packet [10:28:10] (length=34):
<= 04 00 bb 3a 01 40 cf 71 49 7a 4b fa 49 7a 45 7a 09 fa 49 7a 49 7a 49 7a 4f 7a
** received packet [10:28:10] (length=36):
<= 02 00 ba ba 29 00 d0 51 09 7a 49 ea 49 7a 59 7a 09 7a 4d 7a 49 72 49 1a 68 7a
** received packet [10:28:11] (length=38):
<= 00 20 ba ba 09 c0 d1 51 49 7a c9 5a 49 fa 49 3a 09 7a 4b 7a 58 7a 49 7a 69 7a
** received packet [10:28:11] (length=18):
<= 00 00 b2 ba 09 00 d2 71 4d 7b 4b fa 49 3a 59 7a 08 7a 34 7f
** received packet [10:28:12] (length=36):
<= 00 00 bd 3a 11 00 d3 51 0d 7a 48 6a 49 7a 49 7a 48 7a 49 7a 49 7a 49 7a 69 7a
** received packet [10:28:12] (length=12):
<= 00 00 bf ba 08 04 d4 51 09 7a 48 7a 35 7f
** received packet [10:28:13] (length=16):
<= 08 00 ba ba 29 00 d5 71 49 7a 4a 6a 49 7a 51 7a 35 7f
```

Figure 5: The packet log window.

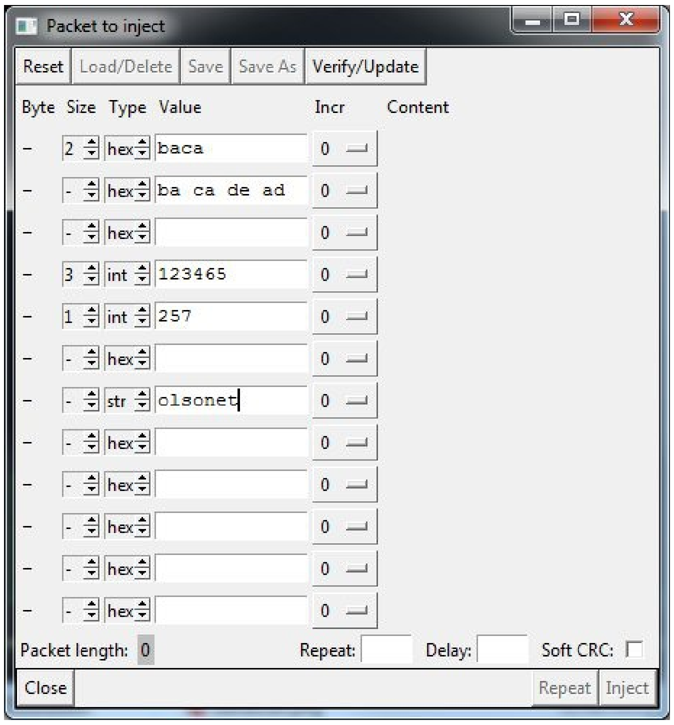
By default, with the MOA (Maximum Of Averages) setting of 1, each RSSI point shifted into the graph is the maximum of 128 consecutive samples collected at the specified “Inter-sample delay”. The 128 samples constitute a package of data arriving from the node. With the MOA setting of 4, for example, each point refers to the maximum of averages of 4-tuples of samples, i.e., 4 consecutive samples are averaged, and the maximum of those averages within the 128-sample package is presented as one point. In the extreme case of MOA=128, all 128 samples arriving in the package are averaged into a single (the degenerate maximum of 1) point.

“Inter-sample delay” from “Scan parameters” applies to the spacing between consecutive samples, but if “Samples to average” is more than 1, then all readings of one averaged sample are collected with no spacing. Note that still 128 samples (each possibly representing multiple averaged readings) contribute to one package, which translates into a point shifted into the graph. By increasing the inter-sample delay, you can slow down the rate at which points are shifted in. For example, at the spacing of 8 milliseconds, the shift-in rate is about 1 second per point (8 x 128 milliseconds), and you still won't miss any activities resulting from “sane” packets (gross length, including sync and CRC, exceeding 10 bytes) sent at 10,000 bps (our standard rate), especially with MOA set to 1. With that setting, the 128 points of the full horizontal span of the graph represent approximately the last 2 minutes of the network's history.

5 Packet log

Note that with the “Band/mode” selection of “Single/rcv”, the program will be receiving packets (those that can be physically received with the given configuration settings of the RF module). Such packets are shown in the “Packet Log” window (see Figure 5). The log is maintained (for a considerable number of packets into the past) even when the window is closed.

Note that the contents of the packet log window can be directed to a file. When you click the “To file” button, the program will open a dialog allowing you to specify the target file to store the log. If the file exists, it will be appended to.



Byte	Size	Type	Value	Incr	Content
–	2	hex	baca	0	
–	-	hex	ba ca de ad	0	
–	-	hex		0	
–	3	int	123465	0	
–	1	int	257	0	
–	-	hex		0	
–	-	str	olsonet	0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	

Packet length: 0 Repeat: Delay: Soft CRC: ☐

Figure 6: The injection window.

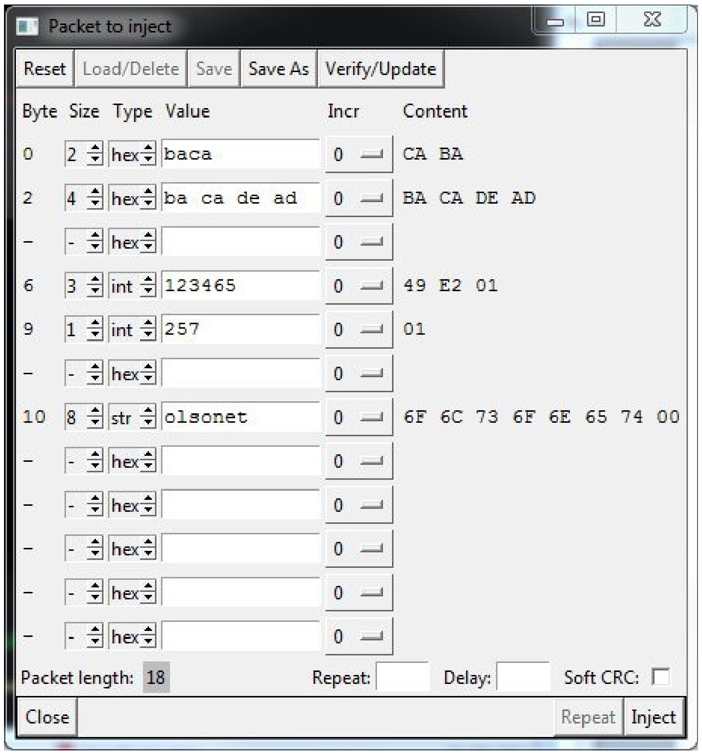
The “Autostart” selection to the left of the “Close” button makes it possible to have the packet log window automatically pop-up whenever the “Single/rcv” scan mode is selected in the main window. In addition to the default setting (“off”), you can chose “on” (with the most natural meaning), and “on+file”. The last option means that in addition to opening the packet log window, the program should also automatically open the last file used to store the log contents (and keep appending new packet data to that file).

6 Packet injection

When you click the “Inject” button, you will open an editor of sorts allowing you to construct the layout of a packet to be injected into the network. While you can build such layouts at any time (and possibly store them in the program's *rc* database), actual injection is only possible during a single-frequency scan (regardless of its parameters). Multiple injection windows can be present at any given time. You can use them to inject multiple (different) packets, possibly in an automated repeat fashion.

A packet is described in terms of fields. Up to 12 such fields can be specified in the window (see Figure 6). The length of a field (size), understood as the length of the packet portion represented by the field, is expressed in bytes. If the length is left unspecified and the value is non-empty, the program will make a guess (which can be corrected manually later).

All filled-in fields contribute to the packet in the order from top to bottom. For example, the layout in Figure 6 consists of 5 fields. Note that you can leave holes (unspecified size and no value); they are ignored and take no space in the packet. If the value is nonempty, but the size is not specified, the field's size will be deduced from the type and the value. If the size is specified, but the value is empty, the value will be set to zero (or to an empty string, if the field's type happens to be “str”).

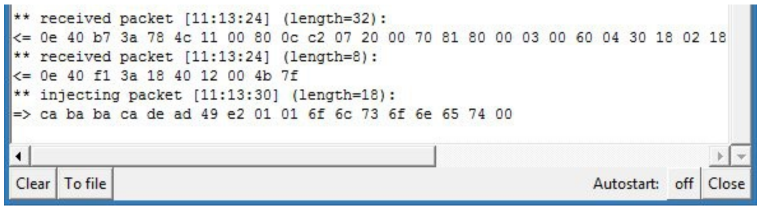


Byte	Size	Type	Value	Incr	Content
0	2	hex	baca	0	CA BA
2	4	hex	ba ca de ad	0	BA CA DE AD
–	-	hex		0	
6	3	int	123465	0	49 E2 01
9	1	int	257	0	01
–	-	hex		0	
10	8	str	olsonet	0	6F 6C 73 6F 6E 65 74 00
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	

Packet length: 18 Repeat: Delay: Soft CRC: ☐

Figure 7: The effect of Verify/Update.

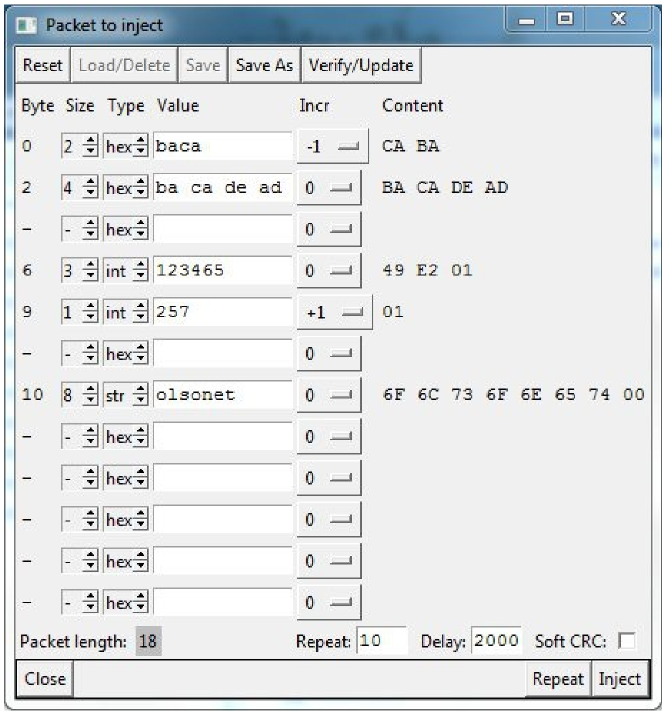
A multi-byte hexadecimal value written as a single number (the first field in Figure 6) is interpreted as little-endian, i.e., its least significant byte will go first. If a hexadecimal value is written as pairs of (hex) digits separated by spaces (e.g., the second field), then each pair stands for one byte, and the bytes go exactly in the written order (from left to right). To see the actual content of the packet, press "Verify/Update". This will cause the fields to be parsed. The unspecified sizes will be calculated and the actual byte content of the packet will be generated and shown (up to 8 initial bytes of every field), as illustrated in Figure 7.



```
** received packet [11:13:24] (length=32):
<= 0e 40 b7 3a 78 4c 11 00 80 0c c2 07 20 00 70 81 80 00 03 00 60 04 30 18 02 18
** received packet [11:13:24] (length=8):
<= 0e 40 f1 3a 18 40 12 00 4b 7f
** injecting packet [11:13:30] (length=18):
=> ca ba ba ca de ad 49 e2 01 01 6f 6c 73 6f 6e 65 74 00
```

Figure 8: An injection report.

Note that the string "olsonet" takes 8 bytes (and includes the zero sentinel). The total length of the packet amounts to 18 bytes.



Packet to inject

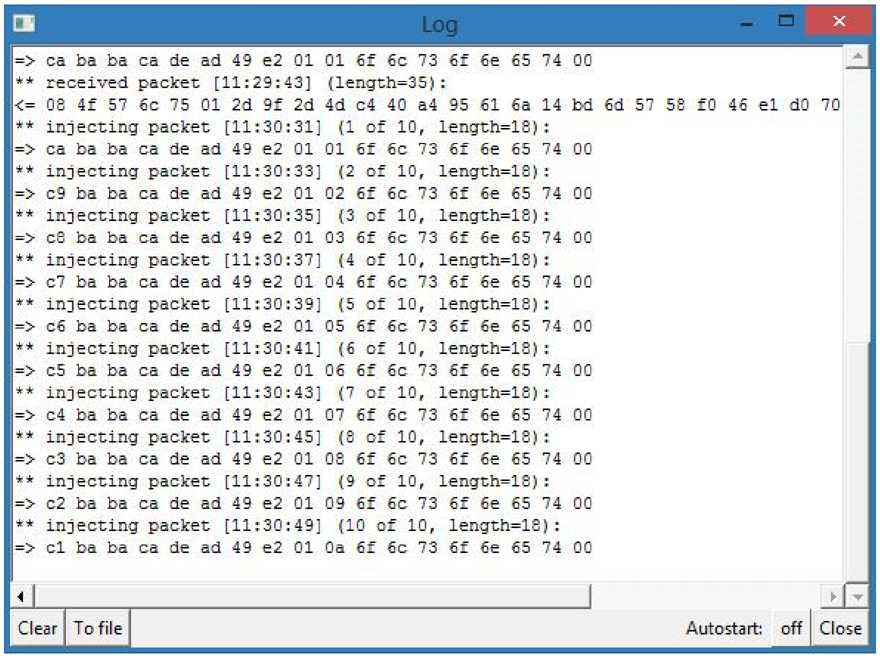
Byte	Size	Type	Value	Incr	Content
0	2	hex	baca	-1	CA BA
2	4	hex	ba ca de ad	0	BA CA DE AD
–	-	hex		0	
6	3	int	123465	0	49 E2 01
9	1	int	257	+1	01
–	-	hex		0	
10	8	str	olsonet	0	6F 6C 73 6F 6E 65 74 00
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	

Packet length: 18 Repeat: 10 Delay: 2000 Soft CRC: ☐

Figure 9: An illustration of incrementing/decrementing.

A manually specified size overrides any ideas the program might have about the field's size based on the type and value. For example, the value 257 in the fourth field has been squeezed into a single byte (truncated) yielding 01. For strings (as well as multiple hexadecimal values, like the one in field 2), when the explicitly specified size is shorter than the value, the trailing bytes of the value (which notion is always clear) are ignored. If the size is larger than required, the trailing bytes of the field will be filled with zeros. Simple numbers are always interpreted as little endian, so basically the same principle applies to them, except that for a negative integer ("int" type) the trailing bytes of an oversized field are filled with the sign extension.

The role the "Incr" selectors is to provide for fields that are automatically incremented or decremented on every injected packet (as a way of differentiating those packets in a systematic way). For example, when you select +1 in the "Incr" widget at the fourth field, the second packet injected from this window will have 02 in its 9'th byte, the third 03, and so on. Only simple numerical values can be incremented/decremented; strings, as well as, multi-part hexadecimal values, cannot.



```
=> ca ba ba ca de ad 49 e2 01 01 6f 6c 73 6f 6e 65 74 00
** received packet [11:29:43] (length=35):
<= 08 4f 57 6c 75 01 2d 9f 2d 4d c4 40 a4 95 61 6a 14 bd 6d 57 58 f0 46 e1 d0 70
** injecting packet [11:30:31] (1 of 10, length=18):
=> ca ba ba ca de ad 49 e2 01 01 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:33] (2 of 10, length=18):
=> c9 ba ba ca de ad 49 e2 01 02 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:35] (3 of 10, length=18):
=> c8 ba ba ca de ad 49 e2 01 03 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:37] (4 of 10, length=18):
=> c7 ba ba ca de ad 49 e2 01 04 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:39] (5 of 10, length=18):
=> c6 ba ba ca de ad 49 e2 01 05 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:41] (6 of 10, length=18):
=> c5 ba ba ca de ad 49 e2 01 06 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:43] (7 of 10, length=18):
=> c4 ba ba ca de ad 49 e2 01 07 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:45] (8 of 10, length=18):
=> c3 ba ba ca de ad 49 e2 01 08 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:47] (9 of 10, length=18):
=> c2 ba ba ca de ad 49 e2 01 09 6f 6c 73 6f 6e 65 74 00
** injecting packet [11:30:49] (10 of 10, length=18):
=> c1 ba ba ca de ad 49 e2 01 0a 6f 6c 73 6f 6e 65 74 00
```

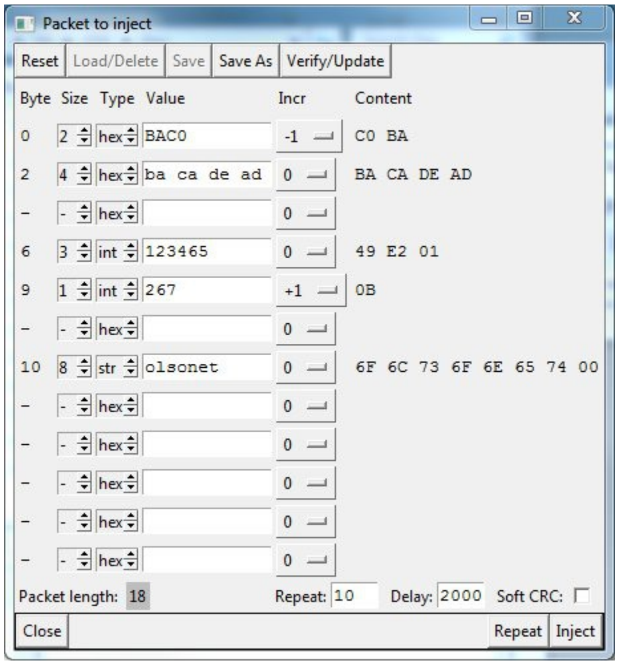
Figure 10: A report from repeat injection.

Whenever you change anything in the configuration of fields, you have to press "Verify/Update" for the change to be reflected in the actual shape of the packet. Any problems with the interpretation of fields will be then diagnosed and reported. This also applies to the two input widgets at the bottom labeled "Repeat" and "Delay", as well as the "Soft CRC" checkbox. When that box is checked, the program will append software CRC (ISO 3309) at the end of the packet before submitting it for transmission. This is only possible if the packet consists of an even number of bytes.

With "Repeat" and "Delay" you can set up a looped transmission of the packet "Repeat" times at "Delay" intervals (expressed in milliseconds). Both those fields must be filled with reasonable integer numbers (and verified-updated) before the "Repeat" button will activate. Generally, the "Inject" button activates when the layout has been verified as correct (there is a packet ready to be injected) and the program is currently executing a single-frequency scan. For the "Repeat" button, the extra condition includes sane contents of the "Repeat" and "Delay" fields.

By hitting "Inject" you inject a single copy of the packet. If the packet log window is open (Figure 5), you will see the packet in the window. For example, for a packet generated according to the layout in Figure 7, you will see a line shown at the bottom of Figure 8.

To see the effect of decrementing/incrementing, modify the layout in the way reflected in Figure 9. Note that the "Repeat" button is now enabled. When you hit it, you will see in the log window ten packets showing up at 2 second intervals (Figure 10). Note how the modified fields differ. Also note that after every injection the inject window reflects the updated values of those fields (see Figure 11).


Packet to inject

Reset | Load/Delete | Save | Save As | Verify/Update

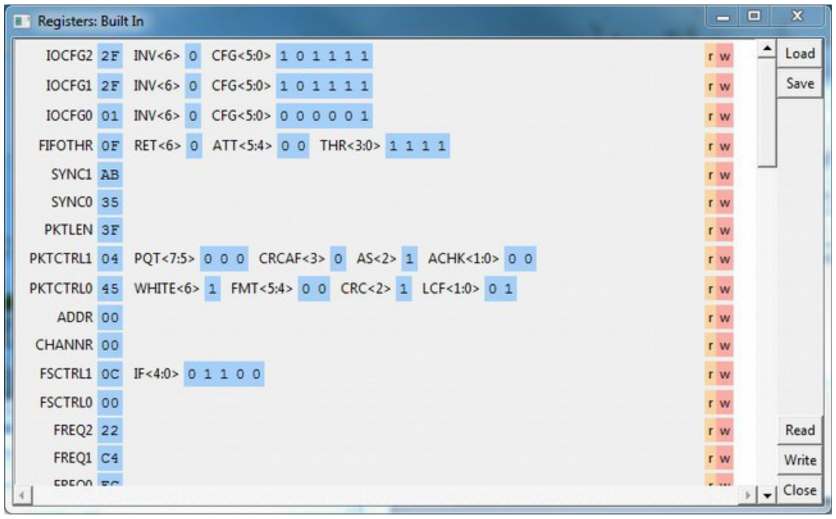
Byte	Size	Type	Value	Incr	Content
0	2	hex	BAC0	-1	C0 BA
2	4	hex	ba ca de ad	0	BA CA DE AD
–	-	hex		0	
6	3	int	123465	0	49 E2 01
9	1	int	267	+1	0B
–	-	hex		0	
10	8	str	olsonet	0	6F 6C 73 6F 6E 65 74 00
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	
–	-	hex		0	

Packet length: 18 Repeat: 10 Delay: 2000 Soft CRC: ☐

Close Repeat Inject

Figure 11: After the repeat injection.

A packet layout can be saved and assigned a name under which it can be later retrieved. The usage of buttons “Load/Delete”, “Save”, “Save As”, as well as “Reset” is obvious.


Registers: Built In

IOCFG2	2F	INV<6>	0	CFG<5:0>	1 0 1 1 1 1						r w
IOCFG1	2F	INV<6>	0	CFG<5:0>	1 0 1 1 1 1						r w
IOCFG0	01	INV<6>	0	CFG<5:0>	0 0 0 0 0 1						r w
FIFOTHR	0F	RET<6>	0	ATT<5:4>	0 0	THR<3:0>	1 1 1 1				r w
SYNC1	AB										r w
SYNC0	35										r w
PKTLEN	3F										r w
PKTCTRL1	04	PQT<7:5>	0 0 0	CRCAF<3>	0	AS<2>	1	ACHK<1:0>	0 0		r w
PKTCTRL0	45	WHITE<6>	1	FMT<5:4>	0 0	CRC<2>	1	LCF<1:0>	0 1		r w
ADDR	00										r w
CHANNR	00										r w
FSCTRL1	0C	IF<4:0>	0 1 1 0 0								r w
FSCTRL0	00										r w
FREQ2	22										r w
FREQ1	C4										r w

Load Save Read Write Close

Figure 12: The registers window.

7 Registers

When you hit the “Show Regs” button in the main window, you will see the “current” configuration of registers (see Figure 12). If you haven't defined any “personal” register sets, there is a default set, named “Built In”, which is selected automatically. If you define and save a set named “Default”, that set will be loaded upon startup replacing the “Built In” set. In any case, the “Built In” (immutable) set can always be referenced and loaded explicitly.

The current register set, be it “Built In”, “Default”, or any other set currently in effect, is written to the device before the commencement of any scan operation. It is obvious that by modifying the register contents you can mess up the node's RF module; however, there is always an easy way out: load the “Built In” set and write it to the node. You don't even have to hit the “Write” button in the registers window, because, as we said, the current set is always written to the node before a scan. In fact, to make sure that the node is in an absolutely sane state, it is reset before the registers are written, so it is pointless to try to modify them otherwise before a scan. Consequently, the “Write” button is probably useless (I may eliminate it in a future version).[1]

Individual registers can be read and written during a scan by clicking the “r”/”w” buttons (they are not really buttons but special labels). Note that some registers are read-only. You will see them at the bottom of the pane, if you scroll it all the way down.

Note that those registers that are logically partitioned into fields have their fields listed and their contents presented in binary (in addition to the hexadecimal content of the complete register). When you hover the mouse over a register/field name, you will see a functional description of the respective item. It may be cryptic in some cases; the manual is cryptic in spots, too, and I do not understand everything.

To modify a register, you can point the mouse to the hexadecimal value (of the full register) or to the binary value (of a field) and click the digit you want to change. A left click on a hex digit will advance it forward, while a right click will push it backward. For a binary digit, either click will flip the bit. Note that such a change is not immediately sent to the node: for that you have to click the small “w” button on the right (or wait until the next scan). You can also edit the register values in order to save the set (under some name) for future reference.

8 Autostart

The “Autostart” checkbox in the main window (see Figure 1) turns on the autostart mode whereby immediately after connecting to the node, the program will commence the (respective) scan (according to the current settings). Note that the status of the “Autostart” checkbox (as well as all other settings) are preserved in the *rc* file (Section 2) for the next invocation of the program. Also note that this (main) “Autostart” function can be combined with the “Autostart” selection in the packet log window. For example, if you also select “on + file” from the log window's “Autostart” menu, and the mode setting is “Single/rcv”, then immediately after connection the program will be showing received packets in the log window and storing them (appending) to the last file associated with the log window. For that to work, such a file must be known, i.e., the *rc* file must include its identity from a previous run.

9 Preconfiguration

The OSS script can be easily preconfigured for special cases, such that the defaults (assumed when no *rc* file is present) differ from the standard ones. The easiest way to do it is to open the standard program, modify the settings as needed, possibly including the registers, and close it, such that the modified settings are written to the *rc* file. If you want to preconfigure registers, make sure to save their new settings under “Default” (Section 7). Then open the OSS script (file *sa.tcl*), as well as the *rc* file, in an editor. Locate the line that starts with set PREINIT … at the very end of the script. Then take the entire contents of the *rc* file (it is a single line of text looking like a Tcl list) and make them a list argument of the set

[1] The praxis has a compilation option disabling the hard reset before a scan.

statement (as illustrated in the code). Make sure to include the external braces as to form a complete list. Finally, replace 0 with 1 in the if statement encapsulating the set statement. Here is how the entire piece of code should look:

```
if 1 {
        set PREINIT { PARAMS {AST 1 LON 1 LOF {} FCE 868.00 BAN
        Single/rcv STA 4 ISD 1 GRS 64 MRS 255 EMA 1 MOA 1 TMX 1}
        REGISTERS {Default {{IOCFG2 47} {IOCFG1 47} {IOCFG0 1}
        {FIFOTHR 15} {SYNC1 171} {SYNC0 53} {PKTLEN 63}
        {PKTCTRL1 4} {PKTCTRL0 69} {ADDR 0} {CHANNR 0} {FSCTRL1
        12} {FSCTRL0 0} {FREQ2 34} {FREQ1 196} {FREQ0 236}
        {MDMCFG4 202} {MDMCFG3 131} {MDMCFG2 3} {MDMCFG1 66}
        {MDMCFG0 248} {DEVIATN 52} {MCSM2 7} {MCSM1 3} {MCSM0
        24} {FOCCFG 21} {BSCFG 108} {AGCCTRL2 3} {AGCCTRL1 64}
        {AGCCTRL0 145} {WOREVT1 135} {WOREVT0 107} {WORCTRL 1}
        {FREND1 86} {FREND0 16} {FSCAL3 169} {FSCAL2 42} {FSCAL1
        0} {FSCAL0 13} {RCCTRL1 0} {RCCTRL0 0} {FSTEST 89}
        {PTEST 127} {AGCTEST 63} {TEST2 136} {TEST1 49} {TEST0
        2} {PATABLE {3 28 87 142 133 204 198 195}}}} }
}
```

The part directly pasted from the *rc* file is shown in bold.