Progetto Monte Ucia

Openwebrx

Speriamo che prima o dopo si ritorni in zona gialla e allora potremo finalmente intervenire su Ucia e riprendere i lavori sul campo. A tale proposito, viste le molte attività che ci aspettano, pensiamo di effettuare una serie di salite nel corso dei prossimi mesi. Quindi, se ci sono volenterosi che hanno voglia di collaborare sono pregati di comunicarlo al presidente. Verranno informati in anticipo delle attività da svolgere e potranno decidere se partecipare alle passeggiate, magari anche solo per rimettersi in forma.

Ed eccoci all'"OPENWEBRX"!! Per prima cosa vorrei citare una frase di un amico radioamatore: "Per fare una buona radio analogica bastano poche decine di transistor, per farne una digitale superiamo abbondantemente il miliardo (contenuti nel processore e nella memoria); non male….., ma questa è la strada".

Come dicevo nel precedente articolo, ho deciso di optare per una installazione da zero. Questo principalmente per due motivi.

Il primo ha a che fare con la sicurezza. Installare un pacchetto preparato e gestito da altri fa sicuramente risparmiare tempo e, probabilmente, il tutto è stato adeguatamente testato sotto molti aspetti, compreso quello della sicurezza. Ma…. chi lo ha messo insieme non conosce l'ambiente in cui io penso di utilizzarlo. D'altra parte io non so quali accorgimenti sono stati adottati a protezione e quali no. Per ripercorrere il lavoro già fatto dal pacchettizzatore perderò un bel po' di tempo e, a meno di contattarlo, non mi saranno chiare tutte le scelte effettuate per mettere in sicurezza il sistema. Per cui preferisco partire da una nuova installazione. Certo….??? ovviamente tutto

questo discorso è proporzionato alle mie capacità ed esperienze.

E qui ci colleghiamo al secondo motivo. Io sono un praticone e imparo molto se, facendo una nuova attività, ci metto dentro il naso. Adottando la "nasite" come metodologia di lavoro, ho pensato di estenderla al resto del team "Ucia". Il gruppetto è composto da persone con background diverso e alcuni di noi non hanno mai affrontato una installazione completa in ambiente Linux. Perché non sfruttare l'occasione per perdere un po' delle paure nel mettere mano al sistema operativo e ai programmi?

Ed eccoci all'installazione, in parte è stata effettuata attraverso memo scambiate tramite Telegram e in parte a quattro mani tramite Teamviewer.

La prima scelta è stata quella dell'hardware. Il Raspberry PI4 4GB sarebbe favorito perché ha una buona potenza di calcolo e tanta memoria. L'idea è di aver potenza di calcolo per poter gestire fino a 4 utenti collegati simultaneamente. Ma si potrebbe ripiegare anche sul PI3 1GB perché consuma relativamente poco (cosa importante su Ucia), a prezzo di ridurre il numero di utenti a 1 o 2.

Dopo aver acquistato i 4 Raspberry, necessari per lavorare in gruppo a distanza, siamo passati all'installazione del sistema operativo Raspbian 10 (denominato Buster). Per l'installazione abbiamo scaricato dal sito raspberrypi.org la versione "2020-02-13-raspbian-buster.zip". Avrei potuto scegliere anche la più leggera "2020-02-13-raspbian-buster-lite.zip" senza l'ambiente grafico, ma considerando che volevo far sperimentare agli amici anche la parte grafica, ho optato per la prima.

A supporto ho scelto il programma "balenaEtcher-Portable-1.5.116.exe" che permette, in ambiente Windows, di trasferire l'immagine del sistema su schedina SD. Una volta effettuata l'operazione tramite il browser di Windows abbiamo aggiunto il file vuoto denominato "ssh" per permettere una successiva connessione al Raspberry anche in assenza di video e tastiera locali.

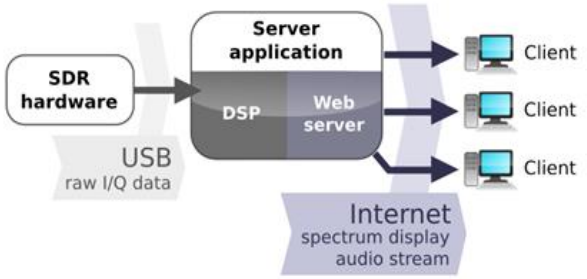
Una volta inserita la SD nell'apposito alloggiamento del Raspberry abbiamo alimentato il sistema, atteso che finisse il riavvio e connesso il nostro nuovo server alla rete tramite cavo ethernet. Per trovare l'indirizzo IP (dato che non tutti erano in grado di accedere al DHCP) abbiamo utilizzato il programma Advanced_IP_Scanner. Per la connessione al server abbiamo utilizzato il software Putty.

Per valutare i consumi in termini di potenza

assorbita ho raccolto i dati che poi riassumerò in una tabella. Per prima cosa ho confrontato PI3 e PI4 accesi senza sistema operativo: Il PI3 assorbe 0,36W, mentre il PI4 2,90W; con il sistema operativo il PI3 1,12W, il PI4 2,06W; con il cavo ethernet connesso alla rete il PI3 1,32W e il PI4 2,42W.

Utilizzando Putty per la connessione in SSH (Secure Socket Shell) abbiamo effettuato la personalizzazione del nostro server: controllato il corretto settaggio dell'ora, con "raspi-config" impostato la password dell'amministratore, il nome del server, la localizzazione e la lingua, l'eventuale rete WIFI, espanso la partizione in modo da occupare tutta la SD. Effettuato il primo riavvio del sistema, abbiamo "upgradato" il sistema all'ultimo aggiornamento, installato un navigatore e editor testuale minimale "MC". Con questa configurazione ho considerato pronto il sistema. Per permettere la sperimentazione di una connessione grafica remota sia da ambienti Windows che Linx, abbiamo installato "xrdp". Tale strumento comunque non ci servirà su Ucia per cui nell'installazione definitiva sarà rimosso. Il server nella sua versione base è, nel complesso, abbastanza sicuro e poco vulnerabile. La verifica lato consumi,

con WIFI attivo, dà 1,27W per il PI3 e 2,40W per il PI4.

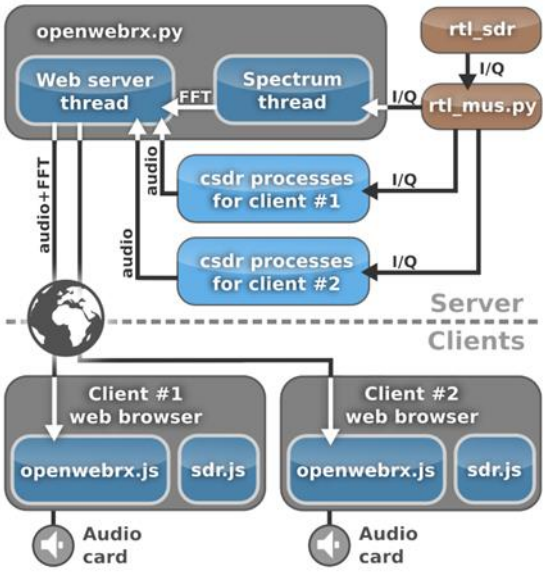


Si passa ora all'OPENWEBRX e quindi iniziamo con un po' di storia e di architettura dell'applicazione. Il software nasce, nel 2013, come tesi di laurea di András Retzler, HA7ILM, e mira ad offrire uno strumento per il radioascolto a basso costo, multiutente (più persone lo possono usare simultaneamente), accessibile da remoto. Inizialmente prevede: di fornire la demodulazione per AM/FM/SSB; di utilizzare un front-end USB dongle basato sul chip RTL2832U; di offrire una interfaccia WEB con possibilità di un "waterfall"; di permettere ad ogni utente connesso di ascoltare, all'interno della banda, il segnale desiderato (frequenza, demodulazione, filtro). Andreas ha manutenuto e ottimizzato il pacchetto per circa 6 anni. Alla fine del 2019 ha annunciato la fine dello sviluppo da parte sua. Successivamente Jackob Ketterl DD5JFK ha preso le redini del progetto e ha introdotto una serie di interventi di ampliamento delle caratteristiche, pacchettizzazione e manutenzione. Il 26/1/2021 Jackob ha rilasciato la versione 0.20.3.

L'architettura iniziale, ancora valida, prevede l'utilizzo di una catena di componenti software. I due blocchi principali sono: l'applicazione server e il front-end in esecuzione sul computer client del utente. Il server è stato implementato in python , un linguaggio di scripting molto flessibile, che ha sicuramente contribuito a ridurre il tempo necessario per sviluppo.

Oltre a implementare il servizio Web, il server genera diversi processi che comunicano, tra loro e con il server core, tramite pipe del sistema operativo (l'uscita di un processo viene inviato all'ingresso dell'altro) e socket TCP . Il programma di input è noto come "**rtl_sdr**" richiamabile da riga di comando. Acquisisce i campioni dei segnali ricevuti dalla chiavetta dal dispositivo RTL-SDR (in genere chiavetta ricevitore USB). Questa soluzione modulare permette di mantenere inalterato il resto del software anche utilizzando differenti ricevitori.

Il componente utilizzato per la distribuzione



dei dati I / Q tra i processi è **rtl_mus.py**, che sta per Server multiutente RTL. Funziona come un TCP server per lo streaming a più client dei dati I / Q non elaborati. Ogni volta che un nuovo client apre la pagina web, il browser avvia una connessione WebSocket, che viene utilizzata per la comunicazione bidirezionale tra il server web e il client. Sul lato server il WebSocket viene utilizzato il modulo rxws.py. A fronte di una nuova connessione WebSocket, il server genera una serie di nuovi processi **csdr** che creano una catena di elaborazione del segnale, che prende l' input da rtl_mus.py ed emette l'audio demodulato. L'audio viene quindi trasmesso, insieme ai dati dello spettro per il diagramma a cascata, dal core WebSocket del server. I dati dello spettro sono emessi dal thread "**Spectrum**", che è comune per tutti i client e quindi ha una sola istanza. L'applicazione front-end in esecuzione sul browser è imple-

mentata in HTML5 e JavaScript: **openwebrx.js** gestisce l'interfaccia utente e il file WebSocket, disegna il diagramma a cascata e trasmette l'audio alla scheda audio utilizzando il Web API audio. Ma prima che l'audio possa essere emesso, alcuni passaggi aggiuntivi di elaborazione del segnale, lato client, sono eseguiti da **sdr.js**.

L'architettura dell'applicazione è stata successivamente arricchita di componenti che permettono: da una parte la commutazione della banda e del ricevitore; dall'altra la demodulazione dei segnali digitali FT8, FT4, WSPR, JT65, JT9, DMR, D-Star, YSF, NXDN, APRS.

Il componente che più mi ha interessato di Openwebrx è il CSDR che è lo strumento per trasformare il segnale I/Q in segnale audio (DSP). Il tutto funziona come una serie di blocchetti da costruzione messi uno in coda all'altro. Ad esempio per demodulare una trasmissione WFM a 100,2 MHz basta eseguire da riga di comando:

```
rtl_sdr -s 240000 -f 100200000 -g 20 - | \
csdr convert_u8_f | \
csdr fmdemod_quadri_cf | \
csdr fractional_decimator_ff 5 | \
csdr deemphasis_wfm_ff 48000 50e-6 | \
csdr convert_f_i16 | \
mplayer -cache 1024 -quiet -rawaudio \
samplesize=2:channels=1:rate=48000 \
-demuxer rawaudio -
```

In questo esempio i dati I / Q raw sono presi dal file output dello strumento rtl_sdr e passati di blocchetto in blocchetto fino ad ottenere il file l'audio demodulato da inviare a mplayer per riprodurlo sulla scheda audio (senza la necessità di OpenWebRX). I passi sono i seguenti:

• rtl_sdr -s 240000 -f 100200000 -g 20 - (il programma RTL-SDR sintonizza a 100,2 MHz con guadagno 20 ed emette campioni I / Q con estensione frequenza di campionamento di 240 ksps) l'uscita di questo programma viene inviata con il comando pipe (" | ") in ingresso al programma

• csdr convert_u8_f (in questo passo si convertono i dati in unsigned a 8 bit campioni in virgola mobile) l'uscita di questo programma viene inviata con pipe in ingresso al programma

• csdr fmdemod_quadri_cf (gestisce un quadri correlatore FM demodulatore, che trasforma il nostro segnale complesso di ingresso in un segnale di uscita reale) l'uscita di questo programma viene inviata con pipe in ingresso al programma

• csdr fractional_decimator_ff 5 (si decima il segnale di un fattore 5, mentre si esegue anche un filtro su di esso per sopprimere la possibile sovrapposizione delle componenti in alta frequenza, si ottiene in uscita un segnale di 48000 sps, che corrisponde al tasso di campionamento della nostra scheda audio) l'uscita di questo programma viene inviata con pipe in ingresso al programma

• csdr deemphasis_wfm_ff 48000 50e-6 (si applica filtro di deenfasi con un tempo costante di 50 μs) l'uscita di questo programma viene inviata con pipe in ingresso al programma

• csdr convert_f_i16 (si convertono i campioni reali in virgola mobile in interi con segno a 16 bit in modo che corrispondano al formato richiesto dalla scheda audio) l'uscita di questo programma viene inviata in ingresso al programma

•mplayer -cache 1024 -quiet -rawaudio (si trasmettono i campioni alla scheda audio).

L'installazione dell'applicazione l'abbiamo fatta:

- scaricando sul Raspberry le librerie di prerequisito, con il comando

sudo apt-get install build-essential git libfftw3-dev cmake libusb-1.0-0-dev nmap git

- bloccando i driver non compatibili blacklist dvb_usb_rtl28xxu, blacklist rtl_2832,

blacklist rtl_2830

- scaricando e assemblando e testando i moduli del pacchetto osmocom per la chiavetta rtl-sdr

git clone git://git.osmocom.org/rtl-sdr.git

rtl_test

- scaricando e assemblando i moduli del pacchetto cdsr sopra descritto

git clone https://github.com/jketterl/csdr.git

- scaricando e assemblando i moduli del pacchetto JS8 Digital Mode

git clone https://github.com/jketterl/js8py.git

- scaricando i moduli del pacchetto soapysdr che contiene i driver per molti tipi di ricevitori sdr

sudo apt-get install libsoapysdr0.6 libsoapysdr-dev soapysdr-tools

sudo apt-get install soapysdr-module-all

- scaricando e assemblando i moduli del pacchetto per connettere i vari driver sdr all'openwebrx

git clone https://github.com/jketterl/owrx_connector.git

- scaricando i moduli del pacchetto per il SOX "Sound eXchange (SoX) - cross-platform audio editing software"

sudo apt-get install sox

- scaricando e assemblando i moduli del pacchetto mbelib "supports the 7200x4400 bit/s codec used in P25 Phase 1, the 7100x4400 bit/s codec used in ProVoice and the "Half Rate", 3600x2250 bit/s vocoder used in various radio systems"

git clone https://github.com/szechyjs/mbelib.git

- caricando e assemblando i moduli del pacchetto digiham per decodificatore per modi digitali tipo DMR

git clone https://github.com/jketterl/digiham.git

- scaricando i moduli della libreria audio I/O

sudo apt-get install portaudio19-dev

- scaricando e assemblando i moduli della libreria per il DSD (Digital Speech Decoder)

git clone https://github.com/f4exb/dsd.git

- scaricando e assemblando i moduli della libreria per la gestione dei protocolli per la voce in digitale

git clone https://github.com/drowe67/codec2.git

- scaricando le librerie prerequisito per il DMR

sudo apt-get install qt5-default libpulse0 libfaad2 libopus0 libpulse-dev libfaad-dev libopus-dev libfftw3-dev

wget https://downloads.sourceforge.net/project/drm/dream/2.1.1/dream-2.1.1-svn808.tar.gz

- scaricando le librerie per l'APRS

sudo apt-get install libasound2-dev

sudo apt-get install direwolf

- scaricando le librerie prerequisito per i modi digitali WSJT-X (FT8, FT4, WSPR, JT65, JT9)

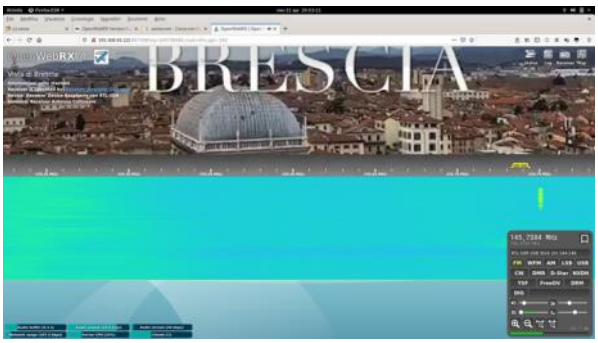
sudo apt-get install asciidoc automake libtool texinfo gfortran libhamlib-dev qtbase5-dev qtmultimedia5-dev qttools5-dev asciidoctor libqt5serialport5-dev qttools5-dev-tools libudev-dev xsltproc

wget http://physics.princeton.edu/pulsar/k1jt/wsjtx-2.1.2.tgz

- scaricando e personalizzando le librerie per openwebrx

git clone https://github.com/jketterl/openwebrx.git -b 0.20.3

Alla fine la metodologia "nasite" ha vinto e alcuni del gruppo sono riusciti ad effettuare



l'installazione in piena autonomia, altri a 4 mani con teamviewer .

Assorbimento del sistema senza chiavetta senza ricevitore: PI3 1,26W, PI4 2,10W

Assorbimento del sistema con ricevitore senza utente: PI3 1,54W, 2,76W. Assorbimento del sistema con ricevitore con 1 utente: PI3 3,6W, 4,62W

Per 1W di differenza penso sia più vantaggioso l'utilizzo del PI4 che offre la possibilità di un maggior numero di utenti.

Adesso abbiamo iniziato la fase di test relativa alla qualità dei ricevitori SDR. I risultati nel prossimo articolo.

I2IPK Tony, I2LQF Fabio,
I2NOS Giuseppe, IZ2DJP Adelio,
IZ2FLY Ernesto