LAUREA TRIENNALE IN SCIENZE E TECNOLOGIE INFORMATICHE

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

ANALISI E PROGETTAZIONE DI UN'ARCHITETTURA PER IL MONITORAGGIO A BASSO COSTO DI CATASTROFI NATURALI

Relazione finale in
Reti Di Calcolatori

Relatore:
Gabriele D'Angelo
Correlatore:
Stefano Cacciaguerra

Presentata da:
Pietro Benini

Sessione Ottobre
Anno accademico 2013-2014

Indice

1 Introduzione

Il monitoraggio ravvicinato di scenari protagonisti di fenomeni ambientali catastrofici, come improvvisi terremoti o violente eruzioni vulcaniche, è reso complicato dalla pericolosità intrinseca di tali ambienti per l'incolumità di operatori e attrezzature.

Generalmente, l'opera di monitoraggio (dalla meteorologia alla sismologia) viene eseguita da stazioni installate in postazioni fisse, nelle quali possono operare lontano da rischi e ove un operatore possa recarsi, senza incorrere in pericoli, per una regolare manutenzione.
Non essendo postazioni a rischio, gli enti che si occupano del monitoraggio, possono investire in strumentazione all'avanguardia [1] (si veda Figura 1.1), costosa ma estremamente precisa, per monitorare attimo dopo attimo l'evolversi di un fenomeno.

Per poter sfruttare efficientemente la strumentazione descritta, occorre una meticolosa installazione, limitandosi a depositare le stazioni di monitoraggio alla roccia invece che ancorarle, la loro precisione sarà vanificata dal rumore nato dallo spostarsi della stessa durante i fenomeni tellurici.

Figura 1.1:Moderno sismometro a tre assi

I dati raccolti, a seconda della situazione, vengono immagazzinati in una scatola nera o direttamente inviati all'ente monitorante. Nessuno dei due approcci può essere impiegato per il monitoraggio ravvicinato, il primo assume che dopo il fenomeno misurato sia possibile raggiungere la stazione per recuperare i dati e ciò non può essere garantito. Il secondo necessita di appoggiarsi a una rete di telecomunicazione, generalmente viene utilizzata la rete cellulare UMTS. In primo luogo la copertura UMTS non è garantita su tutto il territorio nazionale, in secondo luogo, a seguito di eventi catastrofici come terremoti ed eruzioni vulcaniche che vogliamo analizzare, non possiamo assumere che tali reti siano ancora funzionanti.

L'obiettivo di questa tesi è di capire se è possibile realizzare un'alternativa alle attuali stazioni monitoranti, caratterizzata da una facile installazione e da un costo contenuto. Sarà fondamentale che la stazione possa trasmettere a distanza tutto ciò che i suoi sensori registrano, finché non sarà resa non operativa da un guasto meccanico causato dall'ambiente circostante.

2 Descrizione del problema e stato dell'arte

2.1 Introduzione

Il principale problema da affrontare nel monitoraggio ravvicinato di scenari catastrofici, come eruzioni vulcaniche o intensa attività sismica, consiste nel dover installare la strumentazione in luoghi ad elevato rischio per l'incolumità degli operatori e in secondo luogo per gli strumenti stessi. Le fonti primarie di informazioni di attività telluriche (legate a terremoti o attività vulcaniche) sono stazioni fisse[1], ancorate al suolo in luoghi strategici a debita distanza dall'origine. Tali stazioni, attualmente adottate dall'INGV, non sono adatte per il nostro scopo poiché costose, difficili da installare e necessitano di manutenzione regolare. Per ottenere misurazioni ravvicinate dei fenomeni sopra descritti, dobbiamo progettare una stazione monitorante alternativa con i seguenti requisiti:

- Costo contenuto
- Semplicità di installazione
- Nessuna manutenzione
- Basso consumo energetico
- Funzionamento autonomo
- Connettività wireless

Costo contenuto: le stazioni saranno collocate in luoghi ad alto rischio, eserciteranno la loro funzione finché potranno, si avrà la possibilità di recuperarle o di spostarle in luoghi più sicuri dopo il loro posizionamento.
Semplicità di installazione: deve essere un sistema pronto all'uso, gli operatori si dovranno occupare di depositare le stazioni nei luoghi prestabiliti, compito che deve poter essere svolto anche da droni, se se ne presentasse il bisogno.
Nessuna manutenzione: il costo di una stazione non varrà mai il rischio di recuperarla per aggiustarla.

Basso consumo energetico: dobbiamo assumere che non sarà possibile dotare ogni stazione di un allacciamento alla linea elettrica, le pendici di un vulcano non sono dotate di tale struttura mentre collocandosi in uno scenario urbano non è detto che dopo la catastrofe tale rete sia ancora funzionante. A luce di ciò, le stazioni dovranno essere dotate di un'alimentazione a batteria e per tanto parsimoniose nel consumo energetico: la capacità della batteria determina per quanto tempo la stazione sarà attiva e funzionante.
Questo influisce anche sul piano di posizionamento delle stazioni.
La dimensione della batteria influisce sul peso della stazione e quindi sulla trasportabilità, per questo ottimizzare il consumo energetico sarà un obiettivo fondamentale.
Funzionamento autonomo: fintanto che la stazione sarà attiva, spedirà periodicamente dati su ciò che sta osservando senza che qualcuno la piloti dall'esterno.
Deve svolgere tutta una serie di operazioni in completa autonomia. In particolare, potrebbe decidere di cambiare modalità di funzionamento, di campionamento e di utilizzo di dispositivi.
Connettività wireless: la stazione deve poter comunicare su un ponte radio, non potremo disporre di una rete ethernet cablata per trasmettere i dati acquisiti.

Come base per le stazioni abbiamo deciso di utilizzare Raspberry Pi[2] (si veda Figura 2.1), un single-board computer grande quanto una carta di credito, del costo di circa 30€. La sua recente diffusione ha reso possibile progettare e realizzare, con costi contenuti, prototipi di dispositivi e sistemi complessi. È basato su un System-on-a-Chip(Soc) Broadcom BCM2835 nel quale sono incorporati: un processore ARM1176JZF-S (architettura RISC per ISA ARMv6) a 700MHz, una GPU VideoCore IV e 256 o 512 Megabyte di RAM. Il dispositivo non alloggia nessuna memoria di massa ma incorpora uno slot SD; scheda nella quale installare il sistema operativo.
Per poter utilizzare Raspberry Pi è necessario procurarsi un Sistema Operativo sviluppato appositamente per architettura ARM. La Raspberry Pi Foundation fornisce gratuitamente alcune immagini di Sistemi Operativi specifici per questa SoC. Il più diffuso è Raspbian, un porting della distribuzione GNU/Linux Debian per architettura ARM.

Sul sito ufficiale è possibile trovare progetti su Raspberry Pi in differenti campi dell'informatica che vanno dalla riproduzione di contenuti multimediali, alla video sorveglianza o al rilevamento meteorologico. Installando XBMC[3] diventa un riproduttore multimediale per audio e video.
XBMC, abbreviazione di Xbox Media Center, è una piattaforma open source per la riproduzione di contenuti multimediali. Nata per Xbox poi adattata per le varie distribuzioni Linux e sistemi operativi Windows e Microsoft[4].
Può essere usato come client torrent, collegandolo a 2 hard drive può essere configurato come NAS (Network Area Storage) e occuparsi del mantenimento di un archivio dati.

Su Adafruit[5], sito web fondato da Limor Fried, ingengere

dell'MIT, con lo scopo di creare un luogo online per imparare l'elettronica, sono presenti numerosi tutorial per progetti elettronici e informatici basati sull'open hardware e open software. Uno degli esempi innovativi di utilizzo della scheda Raspberry Pi suggeriti è l'utilizzo dello stesso come server proxy Tor[8](in Figura 2.2).

Figura 2.1: RaspberryPi Modello B

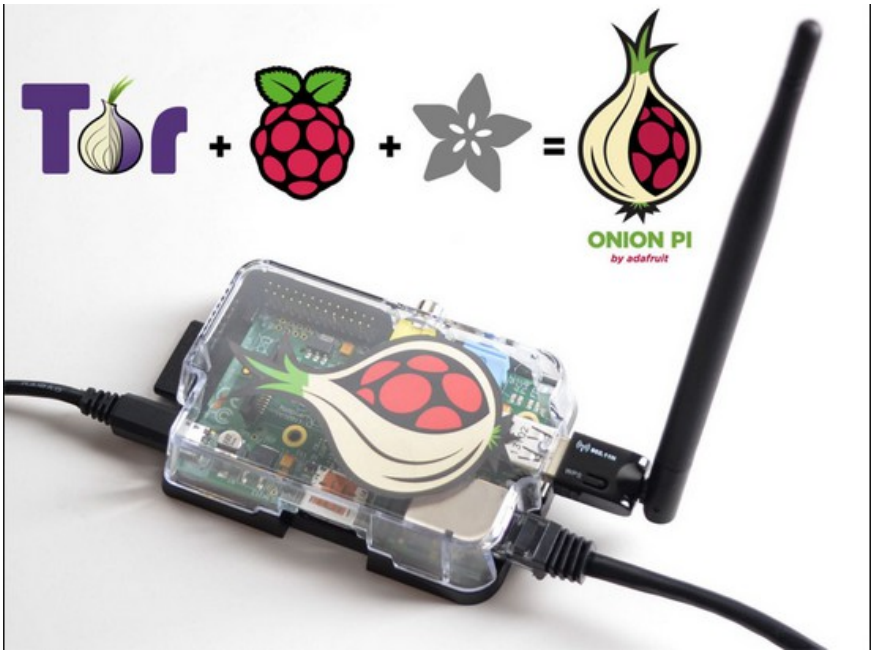

Figura 2.2:Onion Pi

Sta avendo molto successo anche nel campo della Domotica, dove viene utilizzato per controllare impianti elettrici ed elettrodomestici poiché, essendo dotato di sistema operativo Linux,

è possibile sfruttare i vari demoni presenti per gestire le routine domestiche (una su tutte la gestione del riscaldamento), e all'occorrenza dotarlo di web server (es:Apache[6] o Lighttp[7]) per facilitarne l'interfacciamento remoto.

Aggiungendo lo specifico modulo: “Raspberry Pi Camera”[2] in Figura 2.3 (del quale esiste anche la versione a infrarossi), Raspberry può diventare una stazione di videosorveglianza in rete detta IP Camera (Figura 2.4).

Figura 2.3: Raspberry Pi Camera Module


Figura 2.4: IP Camera

Se collegato a sensori, quali: anemometro, igrometro, barometro, Raspberry può diventare una stazione meteo amatoriale[2] (Figura 2.5).

I progetti sopracitati mostrano come tale dispositivo sia versatile ed adattabile a diversi scenari grazie alla possibilità di

collegare alla piattaforma SoC altri dispositivi hardware. In questa ottica, unendo le capacità di videosorveglianza (es progetto videosorveglianza appena citato) alla possibilità di registrare dati provenienti da specifici sensori come barometro, termometro, esposimetro (es progetto stazione meteo) è possibile realizzare una stazione di monitoraggio. Questo dimostra che è possibile realizzare una stazione general purpose ed in particolare adattarla a specifiche esigenze quali il monitoraggio di eventi catastrofici inserendo come sensori ad esempio un accelerometro o giroscopio nel caso si voglia rilevare movimenti tellurici o un sensore di gas zolfo, per misurare l'attività vulcanica.

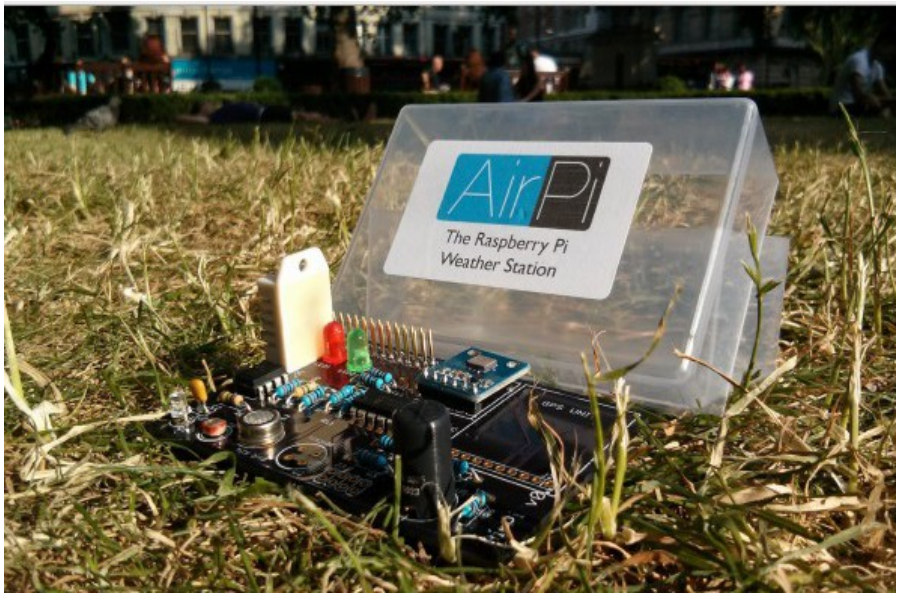

Figura 2.5:Stazione Meteo

2.2 Struttura della rete

Per riuscire ad utilizzare i risultati di tali stazioni in tempo reale è necessario realizzare un'infrastruttura di rete (Figura 2.7) che permetta di collegarle ad un sistema di centrale per il controllo di tali eventi. Le stazioni monitoranti pertanto devono essere collegate mediante un cavo o wireless alla Sala Controllo e condividere un protocollo di comunicazione che permetta di raccogliere tutti i dati e immagazzinare le informazioni ricevute e, se necessario, intervenire da sessione remota per cambiare il comportamento della stazione. Il modello proposto per la comunicazione è un sistema master – slave, dove la Sala di Controllo decide le attività (i.e. master) specificando compiti alle stazioni (i.e. slave) che come agenti autonomi devono portarli a termine. In questo contesto, le stazioni diventano agenti che svolgono ciclicamente una certa routine programmata, in attesa che la Sala di Controllo comunichi specifiche richieste.

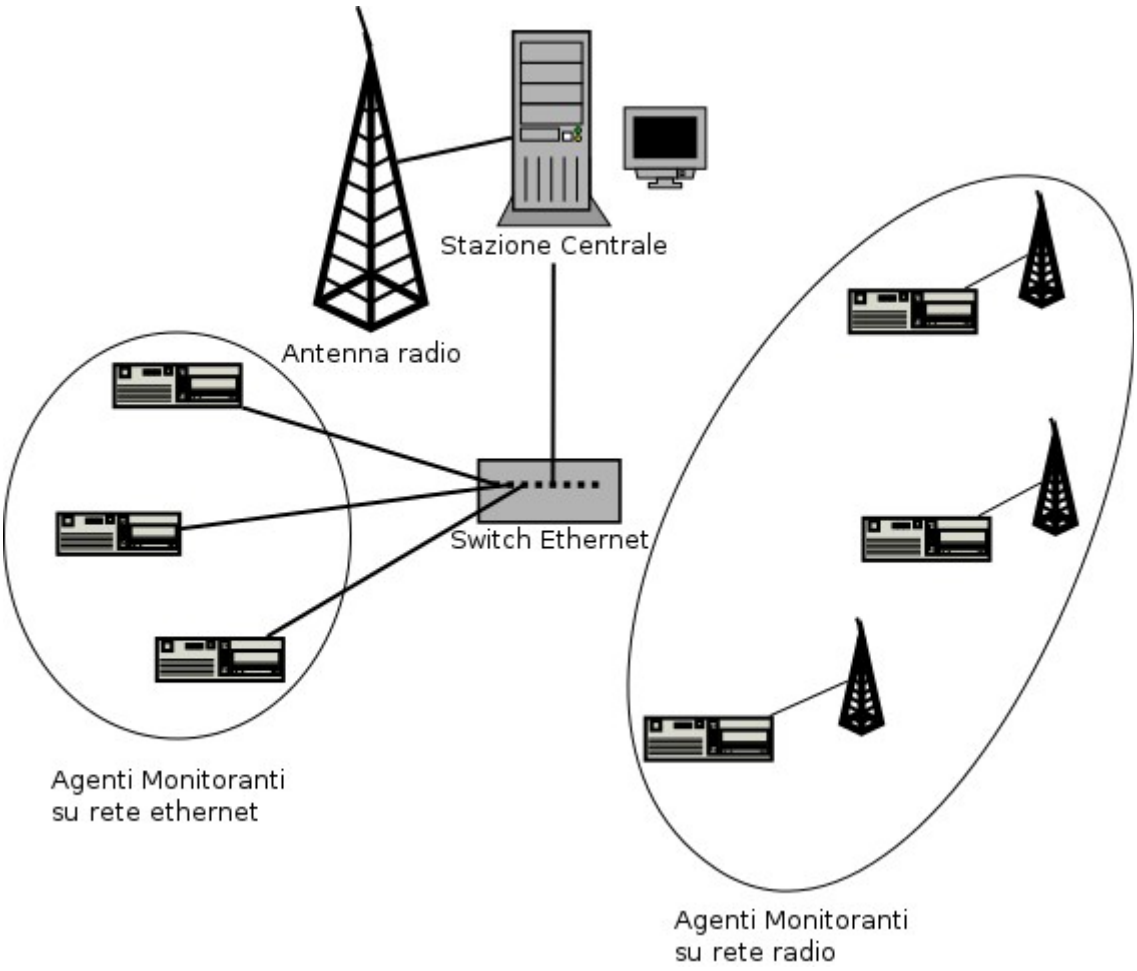

Figura 2.6: schema della rete

2.3 Agente Monitorante

Un Agente[9] è una qualunque entità in grado di percepire l'ambiente che lo circonda attraverso sensori e di eseguire delle azioni attraverso attuatori. Un agente si definisce intelligente se è autonomamente in grado di intraprendere azioni coerenti ai dati che ha acquisito (si veda Figura 2.7). Poiché riteniamo fondamentale avere un riscontro visivo globale dello scenario che stiamo monitorando doteremo tutti gli Agenti di un sensore visivo (webcam). A seconda del particolare scenario nel quale collocheremo ogni agente, aggiungeremo specifici pacchetti di sensori ai quali ci riferiremo con il nome di Moduli.
Tramite un apposito sensore, la stazione potrà misurare le emissioni di gas, mediante un accelerometro capire se la stazione si è spostata. Un giroscopio può rilevare se l'agente è posizionato correttamente o se una scossa lo ha ribaltato, un termometro, misurando la variazione della temperatura, può accorgersi se si sta avvicinando una colata lavica.

Progettare un'architettura modulare ci permetterà di sviluppare una versione base personalizzabile con diverse release al fine di produrre agenti su misura per ogni possibile scenario.

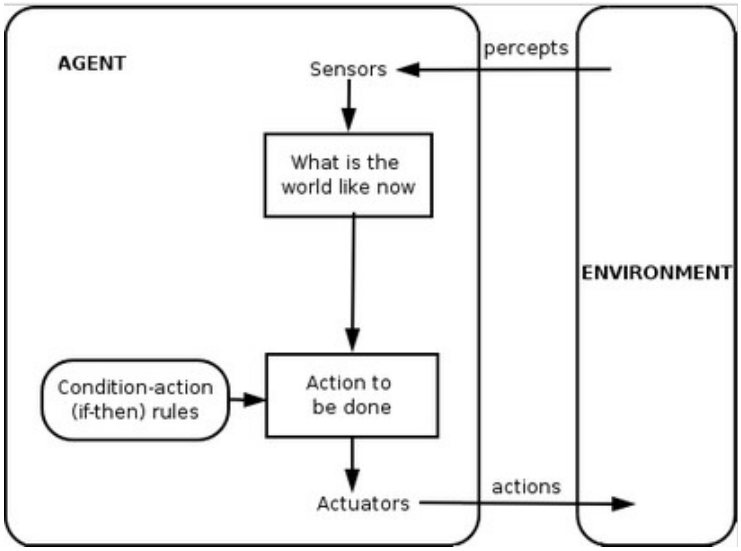

Figura 2.7:Schema logico di un Agente Intelligente[10]

2.3.1 Comportamento

Il paradigma su cui programmeremo gli Agenti per interagire autonomamente con il mondo esterno è chiamato Sense-Think-Act (S-T-A) ed è costituito dal ripetersi ciclico delle tre fasi dalle quali prende il nome.

Le fasi nello specifico:

- Sense: acquisisce informazioni da vari sensori connessi alla stazione
- Think: sulla base dei dati acquisiti da Sense, o da comandi ricevuti dalla sala di controllo, decide se bisogna intraprendere determinate azioni
- Act: attua le decisioni prese da Think

La versione base del nostro Agente, per vulcani e terremoti, seguirà la seguente logica:

Sense: legge l'output dell'accelerometro e lo aggiunge a una coda

Think: legge la coda generata da Sense, se è stata rilevata una scossa incarica Act di scattare una foto

Act: scatta la foto e la invia alla Centrale

La Figura 2.8 mostra un'implementazione sequenziale dei tre stati. Sequenziale poiché l'Agente non può essere contemporaneamente in più di uno stato, nel quale resta fino al completamento delle istruzioni dello stesso. Ne consegue che per tutta la durata di un'attività di Think o di Act, come l'acquisizione di una foto da webcam, l'Agente smetterà di ascoltare l'accelerometro.
È possibile superare questo limite eseguendo le tre fasi su

processi separati, delegando allo scheduler del Sistema Operativo il compito di gestire efficientemente la concorrenza e il parallelismo. Sense accoderà le letture dell'accelerometro in una struttura dati FIFO condivisa con Think. Think leggendo tale coda spedirà alert al processo Act ogni volta che individuerà una scossa. Si veda figura 2.9.

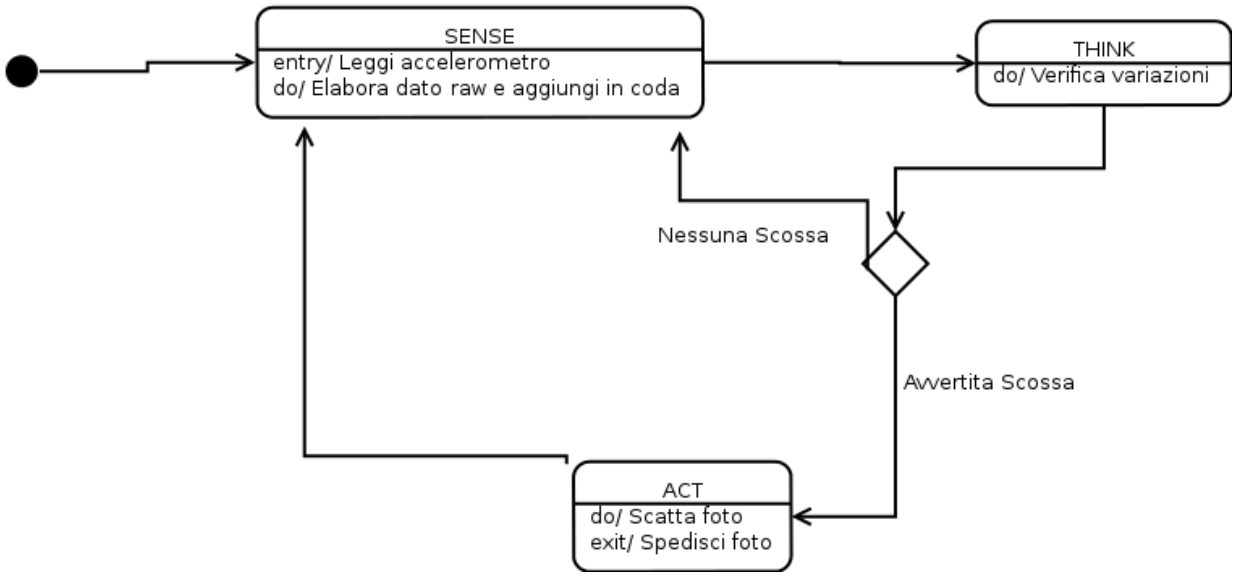

Figura 2.8: Diagramma degli stati di un Agente Monitorante sequenziale

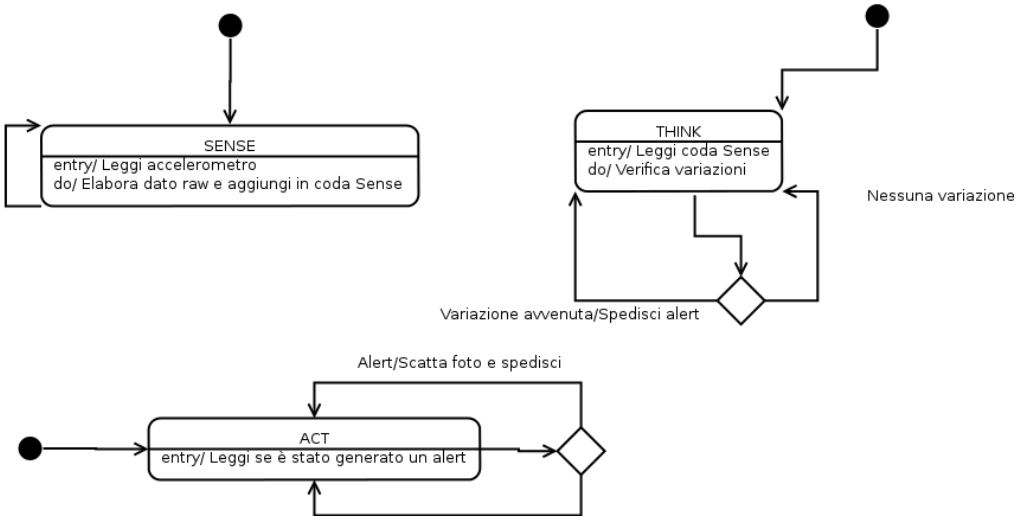

Figura 2.9:Diagramma degli stati di un Agente Monitorante parallelo

2.4 Collegamento

Gli agenti sopra descritti comunicheranno tra di loro attraverso collegamenti cablati o wireless come mostrato precedentemente in Figura 2.1. Analizzeremo ora tali tecnologie per valutare vantaggi

e svantaggi delle diverse soluzioni. Come protocollo cablato abbiamo considerato esclusivamente il collegamento Ethernet poiché la scheda Raspberry Pi è dotata di specifica interfaccia di rete ed è attualmente il sistema LAN cablata più diffuso. Il limite di Ethernet è la complessità di realizzare un'infrastruttura su area metropolitana.
Per aree al di sotto dei 100 metri è possibile collegare le stazioni ad un nodo centrale stendendo un unico cavo per ognuna di esse, per aree maggiori è necessario predisporre dei ripetitori[11].

RETI RADIO
Radio è la famiglia tecnologie che utilizzano onde radio (radiazioni elettromagnetiche di frequenza compresa tra 0 e 300GHz) per trasmettere informazioni a distanza e tra i loro impieghi più diffusi si ricordano: lo stream audio/video, la telefonia mobile e il Radar (Radio Detection And Ranging). Un collegamento radio ci permetterà di contattare stazioni che per motivi fisici o tecnici non potranno essere collegate alla rete fisica sia alla corrente elettrica. Gli agenti wireless saranno stazioni monitoranti collocate, a mano o depositate, in punti strategici dello scenario sotto monitoraggio e per tanto dovranno essere pronte all'uso senza nessuna regolazione fisica sul posto.

Le tecnologie radio più diffuse, legate all'informatica, sono: Bluetooth, WiFi e WiMAX[12].
Riteniamo che nessuna delle tre sia adeguata per i motivi che andremo ad elencare.
Bluetooth ha una copertura e una larghezza di banda eccessivamente ristrette.
WiFi, utilizzando antenne omnidirezionali, pur avendo una copertura maggiore di Bluetooth, non copre un'area adeguatamente grande.
L'utilizzo di antenne direzionali non è stato considerato poiché necessita un puntamento manuale.
WiMAX permette di coprire aree di diversi chilometri ma il suo funzionamento continuato richiede un allacciamento alla rete elettrica. La nostra assunzione è che ovunque sia presente un impianto elettrico attivo, sia altrettanto presente una rete internet cablata.

A luce delle valutazioni espresse sopra, abbiamo deciso di sperimentare come livello di collegamento i moduli:RF Transceiver 431 470MHz, in Figura 2.10, per comprendere se può essere una valida alternativa.

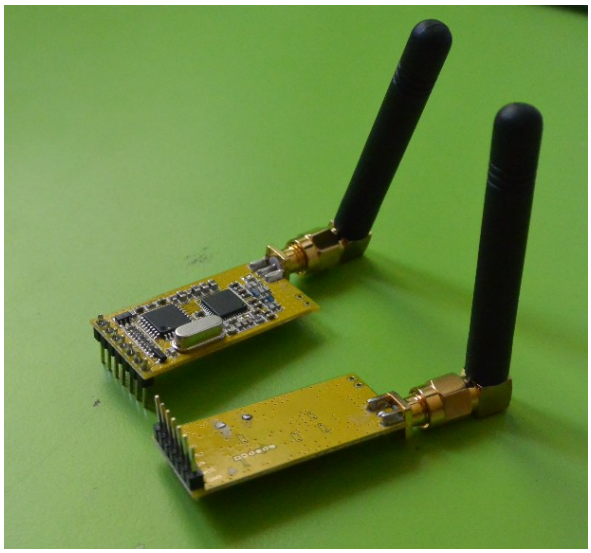

Figura 2.10: coppia di moduli Transceiver

Nei capitoli seguenti sarà illustrato come abbiamo realizzato la rete descritta su un'architettura cablata (Capitolo 3) e la sostituzione della rete cablata con wireless (Capitolo 4).

3 Soluzione cablata

3.1 Introduzione

Data la complessità dello scenario, la soluzione adottata è stata realizzata attraverso due fasi. La prima fase prevede la costruzione di una stazione autonoma monitorante per acquisire dati tramite sensori. La seconda fase prevede lo sviluppo di un protocollo di comunicazione su rete IP delle stazioni monitoranti con la Sala di Controllo.
Nella terza abbiamo studiato le implicazioni della rete fisica sul nostro sistema sala centrale e stazioni monitoranti. Queste implicazioni riguardano la distanza di trasmissione ed il corrispettivo consumo di energia durante la comunicazione tra stazioni autonome monitoranti distribuite su un territorio impervio e non cablato e la Sala Centrale.

3.2 Realizzazione di una stazione monitorante autonoma

Sebbene nel capitolo precedente si è parlato di utilizzare un accelerometro come sensore base per la stazione monitorante, abbiamo scelto di iniziare le sperimentazioni utilizzando un sensore di temperatura per semplicità in quanto fornisce segnali più semplici da elaborare.

3.2.1 I sensori

Un sensore[13] è un dispositivo che permette di trasformare una grandezza fisica in segnale elettrico. Un termistore, per esempio, (Figura 3.1) è un resistore il cui valore di resistenza varia in maniera lineare e significativa con la temperatura.

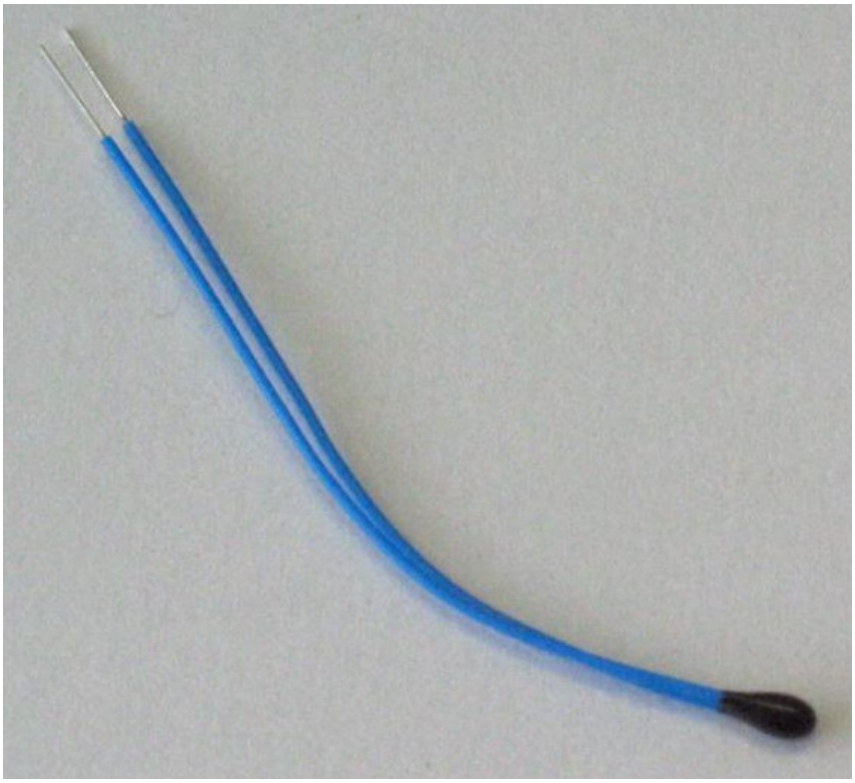

Figura 3.1:Termistore

Fornendo una tensione *v* a un capo e misurando la corrente *i* che lo attraversa è possibile risalire alla resistenza *r* del termistore grazie alle leggi di Ohm:

$$R = \frac{V}{I}$$

e dal valore di resistenza ottenere quindi la temperatura.
Il primo termometro da noi utilizzato è il transistor TMP36[14] fornito nel kit base di Arduino (Figura 3.2), questo tipo di termometro ottiene in input una tensione di 5V e a seconda della temperatura fornisce in output una tensione diversa.

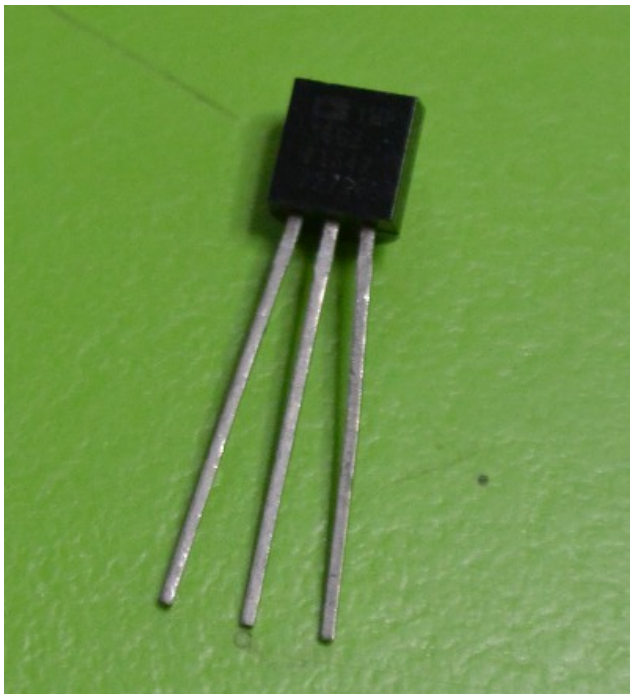

Figura 3.2:Sensore analogico TMP36

Per convertire in maniera automatica l'output del sensore lo abbiamo collegato al micro controllore Arduino Uno (Figura 3.3)

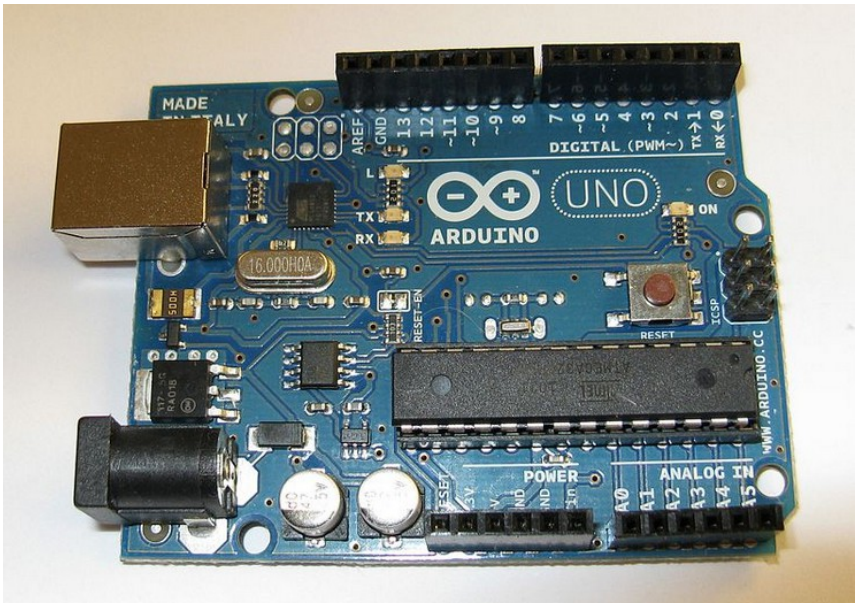

Figura 3.3:Arduino Uno

Arduino[15] è una piattaforma open-source, basata sul micro controllore ATMEGA328. Tramite Arduino è possibile creare prototipi di circuiti elettronici come quello di cui abbiamo bisogno. Abbiamo scelto Arduino perché è possibile programmare il suo comportamento ad alto livello con un linguaggio simile al C. Dopo aver cablato il circuito si collega Arduino al Computer mediante il cavo USB e utilizzando l'Arduino IDE abbiamo scritto e compilato il

seguente sorgente:

```
//DEFINIZIONI DELLE COSTANTI
const int sensorPin=A0;    //IMPOSTIAMO IL PIN ANALOGICO 0 COME PIN DI INGRESSO

//SET UP È LA FUNZIONE PER LE CONFIGURAZIONI INIZIALI
void setup(){
  Serial.begin(9600);         //ISTANZIAMO UNA PORTA SERIALE A 9600 BAUD

}

//DOPO LE CONFIGURAZIONI INIZIALI IL MICRO CONTROLLORE ENTRA IN LOOP
void loop(){
  int sensorVal=analogRead(sensorPin);       //LETTURA DEL TERMOMETRO*
  Serial.print("Sensor Value: ");            //SERIAL PRINT STAMPA SULLA PORTA SERIALE
  Serial.print(sensorVal);

  float voltage=(sensorVal/1024.0)*5.0;      //CONVERSIONE IN VOLT  DEL VALORE MISURATO

  Serial.print(", Volts: ");
  Serial.print(voltage);

  Serial.print(", Degrees: ");
  float temperature =(voltage-0.5)*100;      //CONVERSIONE DEL VOLTAGGIO IN GRADI CENTIGRADI
  Serial.println(temperature);
}

*l'impulso letto da Arduino sul pin A0 viene interpretato come numero compreso tra 0 (assenza di tensione) e
1024 (5V di tensione)
```

Arduino IDE[16] è dotato di un'interfaccia seriale con la quale leggere l'output di Arduino.

Per l'acquisizione di immagini è stato utilizzato il Raspberry Camera module, specifica videocamera per Raspberry descritta nel Capitolo 2. Raspberry si interfaccia al camera module con le specifiche chiamate a sistema: *raspistill*[17] (per snapshot) e *raspivid* (per registrare video).

3.2.2 Programmazione della Stazione Monitorante

Per realizzare il software della stazione monitorante, abbiamo deciso di utilizzare Python[18] come linguaggio di programmazione per i seguenti motivi:

- è un linguaggio interpretato e per tanto non deve essere compilato
- è il linguaggio di programmazione più diffuso su Raspberry
- impone di scrivere codice pulito e ordinato, ad esempio per le regole rigide sull'uso dell'indentazione[19]
- è facile da scrivere e debuggare

Delle versioni disponibili di Python è stata scelta la 2.6 perchè permette una lettura più semplice della porta seriale rispetto alle versioni 3.*.

Nei prossimi paragrafi illustreremo i progressi fatti durante la sperimentazione.

3.2.3 Versione 0

Il primo prototipo di stazione realizzato è composto da una scheda Raspberri Pi alla quale sono state collegate Il modulo Camera e Arduino (Figura 3.4), configurato come spiegato nel paragrafo 3.2.1. Il primo problema che si è presentato è che la porta USB del Raspberry non riesce ad alimentare correttamente Arduino col risultato di ottenere in output misurazioni inesatte. A luce di ciò abbiamo dovuto dotare Arduino di un'alimentazione dedicata a 9 volt.

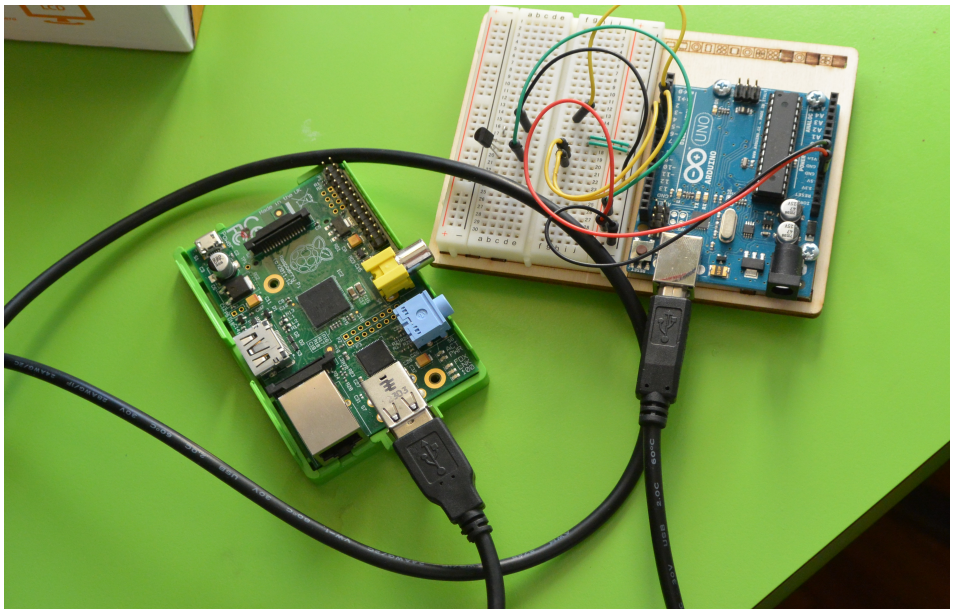

Figura 3.4:Arduino e Raspberry

La versione 0 della stazione alterna sequenzialmente le fasi Sense, Think e Act con i seguenti scopi:

- Sense: scatta una foto, misura e stampa la temperatura dell'ambiente
- Think: ruota la foto di 180 gradi
- Act: copia la foto nella cartella: “/var/www/ProgettoStazionePhoto”

```
#main del progetto, eseguire con Python 2.*

import camera
import time
import letturaTermometro as Termo

def sense():
        print("sense")
        camera.snap()
        TmP=Termo.get_Temp()
        print(“TmP”)

def think():
        print(“think”)
        os.system("convert foto/lastlap.jpg -rotate 180 foto/lastlap.jpg")
def act():
        print("act")
        os.system("sudo cp foto/lastlap.jpg /var/www/ProgettoStazionePhoto")

while 1:
        sense()
        think()
        act()

        time.sleep(5)
```

“letturaTermometro” e camera, sono moduli Python da noi scritti. Per rendere il codice più ordinato e leggibile abbiamo deciso di scrivere tali funzioni su file separati.
LetturaTermometro.py:

```
import serial

def get_Temp():
        ser = serial.Serial('/dev/ttyACM0', 9600)
        while 1:
                Data=ser.readline()
                Temp=Data[41:46]        #della stringa generata da Arduino ritagliamo
                                        #la sottostringa compresa  tra i caratteri
                                        # 41 e 46 che contengono la temperatura
                                        #misurata
                try:
                        float(Temp)
                        print(Temp)
                        break
                except:
                        print("")
        return Temp
```

Camera.py

```
def catch_photo():

    global frameCount

    imageNumber = str(frameCount).zfill(7)
    os.system("raspistill -t 5 -o foto/image%s.jpg -w 640 -h 480"%(imageNumber))
    os.system("cp foto/image%s.jpg foto/lastlap.jpg"%(imageNumber))
    frameCount+=1
```

3.2.4 Versione 1: Sense Think Act su Thread paralleli

La Stazione Monitorante realizzata rispetta le tre fasi (Sense, Think e Act) che ci siamo prefissi, tuttavia non in modo efficiente. Quando viene richiamata il sistema operativo per attivare la videocamera, il programma si ferma in attesa che venga terminato lo scatto, il tutto impiega qualche secondo. Se per esempio avessimo la necessità di contattare da remoto la stazione durante la fase Think, il funzionamento della Stazione Monitorante verrebbe compromesso per un tempo indeterminato. Per eliminare questa problematica abbiamo deciso di lanciare Sense Think e Act su tre thread separati.
Abbiamo pensato che potesse essere utile, per migliorare la leggibilità del programma, scrivere i codici delle tre fasi ciascuna in un file specifico.
Infine abbiamo sostituito “leggiTermometro.py” con “arduino.py” aggiungendo funzionalità di verifica.
I thread sono stati implementati attraverso il modulo “thread”[20].
Main.py

```
import time
import thread
import sense
import think
import act

while 1:

    try:
        thread.start_new_thread(sense.get,())

    except:
        print "Errore:non sono riuscito a far partire il tred Sense"
    try:
        thread.start_new_thread(think.get,())

    except:
        print "Errore:non sono riuscito a far partire il tred Think"
    try:
        thread.start_new_thread(act.get,())

    except:
        print "Errore:non sono riuscito a far partire il tred Act"

   time.sleep(2)
```

Arduino.py

```
import serial
import ast
import time

def get_Dati():

	nTentativi=0
	while nTentativi<5:
		try:
			ser = serial.Serial('/dev/ttyACM'+str(nTentativi), 9600)
			print("Connesso alla porta: /dev/ttyACM"+str(nTentativi))
			break
		except:
			print("Errore di connessione,riprovo")
			nTentativi+=1
	if(nTentativi<5):
		lung=0
		print("START")

		while lung!=18:
			try:
				Dati=ser.readline()
				lung=len(Dati)

			except:
				print("errore di lettura,riprovo")

		print("STOP")
		Dati=Dati[0:-2]
		Dati=ast.literal_eval(Dati)
		Dati['Time']= time.strftime("%X")


		return Dati
	else:
		print("Porta non trovata")
```

3.2.5 Versione 2: Tutte STA in un unico modulo

La versione 2 raggruppa tutte le funzioni di Sense, Think e Act in un unico modulo nominato “myModule.py”. Sense legge ogni secondo la temperatura ambiente e la aggiunge ad una lista, Think verifica se è aumentata di una certa soglia (ad esempio di 2 gradi negli ultimi 10 secondi). In caso affermativo notifica un allarme. La lista creata da Sense, a regime, conterrà le ultime 60 misurazioni di temperatura, ad ogni aggiunta, la temperatura più vecchia sarà eliminata e salvata su un file.
La parte Act non è stata implementata, il nostro obiettivo era di progettare una struttura di software funzionale e da cui partire per lo sviluppo vero e proprio, non un prodotto finito.

Start.py

```
import time
import threading
import myModule

senseTh = threading.Thread(target=myModule.Sense,)
thinkTh = threading.Thread(target=myModule.Think,)

senseTh.start()
thinkTh.start()
```

MyModule.py

```
import time
import arduino
import camera
import threading

L=[]


def Sense():
        photoTh = threading.Thread(target=camera.catch_photo,)
        photoTh.start()
        ser=arduino.connetti()
        frameCount=0
    while 1:
                if(len(L)>=60):
                        L.pop(0)#DA SALVARE SU UN FILE


                Dati=arduino.leggiArduino(ser)
            if Dati['Time'][-2:]=="00":
                        photoTh.lock.release()
                L.append(Dati)
            print(Dati)



def Think():
    while 1:

            if len(L)>=10:
                        if float(L[-1]['Temp'])-float(L[-10]['Temp'])>=2:
                                print("ALLARME")##DA IMPLEMENTARE

                time.sleep(1)
```

3.3 Sviluppo di un protocollo di comunicazione su rete IP

L'obiettivo della seconda fase è stato di progettare la sala di controllo e di gestire le comunicazioni con le stazioni sulla rete Ethernet dell'INGV.

Secondo noi, la Stazione Centrale dovrà fornire i seguenti servizi:

- aprire un socket TCP[21] (Trasmission Control Protocol) sulla quale può essere contattata dalle stazioni. Abbiamo scelto TCP come protocollo a livello di trasporto poiché a differenza di UDP (User Datagram Protocol) fornisce un servizio affidabile e orientato alla connessione[22].
- elencare le stazioni collegate
- fornire tool per facilitare le connessioni con le stazioni utilizzando ssh[23] e sftp[24], ssh è un protocollo di comunicazione a livello applicazione che permette di aprire sulla macchina locale un terminale verso una macchina remota utilizzando un canale cifrato, sftp è un protocollo, come ssh a livello di applicazione, nato per lo scambio di file su Internet, a differenza del protocollo ftp classico, sftp utilizza una connessione cifrata ssh
- notificare se una stazione è silenziosa da troppo tempo

Per realizzare ciò abbiamo realizzato una suite di programmi Python con interfaccia a riga di comando, ognuno dei quali assolve uno specifico compito tra quelli sopracitati.
Per lanciarli parallelamente abbiamo realizzato un main che, tramite il modulo subprocess, apre ogni istanza dei programmi in una shell specifica.
Abbiamo deciso di passare dalla Python2 a Python3. Le due versioni presentano diverse incompatibilità, volendo progettare le basi di un progetto da sviluppare nel tempo abbiamo preferito la più recente.

```
#!/usr/bin/python3

import os

import subprocess


pres =subprocess.Popen(['gnome-terminal -e ./PresentazioneCentrale.py'],shell=True)

elenco=subprocess.Popen(['gnome-terminal -e ./elencaStazioni.py'],shell=True)

ssh=subprocess.Popen(['gnome-terminal -e ./ssh.py'],shell=True)

sftp=subprocess.Popen(['gnome-terminal -e ./sftp.py'],shell=True)

cAggiornamenti=subprocess.Popen(['gnome-terminal -e ./controllaAggiornamenti.py'],shell=True)
```

Seguendo l'ordine con cui il main attiva i processi, il primo ad essere caricato è: “SalaControllo.py”[25].
SalaControllo è un server Python in ascolto sulla porta 50001. Ogni connessione che riceve viene spostata su nuovo socket in un thread dedicato. Il thread aggiunge la stazione alla lista delle stazioni attive e la salva sul file: “listaConessioni”. Salva in locale una copia del file “info” contenente tutte le specifiche di sistema della stazione e chiude infine la comunicazione. La shell di PresentazioneCentrale permette di monitorare il susseguirsi delle operazioni svolte dal server fornendo messaggi di avvenuta connessione.

```
#!/usr/bin/python3
import socketserver, time, base64# get socket server, handler objects
import os
myHost = ''                         # server machine, '' means local host
myPort = 50001                      # listen on a non-reserved port number

ListaStazioni=[]

def checkfile(data):
        global ListaStazioni
        flag=1
        if ListaStazioni==[]:
                ListaStazioni.append(data)
        else:

                for t in ListaStazioni:
                        if t[0]==data[0]:
                                ListaStazioni.remove(t)

                ListaStazioni.append(data)
                ListaStazioni.sort()
        f=open('listaConnessioni','w')
        for t in ListaStazioni:
                print(str(t))
                f.write("".join(str(t)))
                f.write("\n")
        f.close()

def now():
        return time.ctime(time.time())
class MyClientHandler(socketserver.BaseRequestHandler):
        def handle(self):                                       # on each client connect
                print(self.client_address, now()) # show this client's address

                while True:                                     # self.request is client socket
                        data = self.request.recv(1024)   # read, write a client socket

                        if not data: break

                        checkfile((data.decode(),self.client_address[0],now()))
                        reply="ok"
                        self.request.send(reply.encode())
                        nome=data.decode()
                        print(nome)

                        data = self.request.recv(1024)
                        if not data: break
                        print(data.decode())
                        if data.decode()=='scarica':

                                os.system("scp
pi@"+self.client_address[0]+":ProgettoStazione/4Versione/info info/"+nome)
                        reply="ok"
                        self.request.send(reply.encode())
                self.request.close()

# make a threaded server, listen/handle clients forever
myaddr = (myHost, myPort)
server = socketserver.ThreadingTCPServer(myaddr, MyClientHandler)

server.serve_forever()
```

Elenca stazioni legge il file: “listaStazioni” e lo mostra a video. Ad ogni stazione assegna un numero progressivo, a partire da 1.

```
#!/usr/bin/python3
import os
import time

c=1
f=open("listaConnessioni",'r')

L=[]
while(1):
        os.system("clear")
        print("ELENCO DELLE STAZIONI COLLEGATE")
        for t in open("listaConnessioni").readlines():
                print(str(c)+"-"+t)
                c+=1
        c=1
        time.sleep(1)
```

Ssh.py e sftp.py sono programmi che consentono di aprire rispettivamente una connessione ssh e sftp verso le stazioni in maniera automatica. Inserendo nel programma il codice della stazione che vogliamo contattare, esso apre ssh o sftp in una nuova shell. Inserendo 0 il programma avvia il programma:cssh (clustered ssh).
Clustered[26] ssh apre una console di amministrazione e un'interfaccia xterm a linea di comando con una sessione ssh verso ogni host specificato, nel nostro caso verso tutte le stazioni che hanno contattato almeno una volta la sala di controllo. I comandi scritti nella console di amministrazione vengono replicati su ogni host. Allo stesso tempo è possibile controllare ogni host mediante la sua specifica finestra.

```
#!/usr/bin/python3
import os
import time
import subprocess

c=1
f=open("listaConnessioni",'r')
ip=-1


while(1):
        L=[]
        os.system("clear")
        print("SU QUALE STAZIONE FARE SSH?")
        while(ip==-1):
          val=input()
          try:
                ip = int(val)
          except ValueError:
                print("Non e' stato inserito un numero")

        if ip>=0:
                for t in open("listaConnessioni").readlines():
                        app=t[1:-2]
                        app=app.split(',')
                        L.append(app[1][2:-1])
                print(L)
                if ip>0:
                        os.system("xterm -e ssh pi@"+L[ip-1]+" &")
                else:
                        stazioni=""
                        for item in L:
                                stazioni=stazioni+"pi@"+item+" "
                        os.system("xterm -e cssh "+stazioni+ "&")
        time.sleep(2)
        c=1
        ip=-1
```

ControllaAggiornamenti ogni 10 secondi analizza i file contenuti nella cartella “info”. Tale stazione contiene i file con le informazioni di sistema delle stazioni che hanno contattato la Sala di Controllo almeno una volta, e la data di creazione di ogni file coincide con il momento del collegamento.
Se il file risale ad almeno 5 minuti prima del momento in cui è stato eseguito il controllo, il programma notifica a video un errore.

```
#!/usr/bin/python3
import os, time

while True:
	L=[]
	os.system("clear")
	print("ALLARME STAZIONI")
	for file in os.listdir("info/"):
		L.append(file)
	print(L)
	for t in L:
		tempo=int((time.time()-os.stat('info/'+t).st_mtime)/60)
		if tempo>5:
			print("non si ricevono aggiornamenti dalla stazione "+t+" da
"+str(tempo)+" minuti")
	time.sleep(10)
```

Per regolamentare l'attività delle stazioni, abbiamo deciso di delegare il compito al demone cron. Cron[27] è un demone che ha il compito di svegliarsi ogni minuto ed eseguire ogni programma che è stato programmato per quel momento. Il principale file di configurazione di cron è: /etc/crontab che contiene la tabella con l'elenco delle operazioni periodiche generali da eseguire nel sistema. Abbiamo aggiornato tale file con l'istruzione per lanciare ogni minuto, i seguenti due programmi:

```
#!/bin/bash


raspistill -o /home/pi/ProgettoStazione/4Versione/foto/$(date +'%Y-%m-%d-%H-%M').jpg -w 640 -h 480 --rotation 180
```

```
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = ('192.168.103.52', 10000)

sock.connect(server_address)
try:
    f=open('file','r')
    # Send data
    message = f.read()
    print >>sys.stderr, 'sending "%s"' % message
    sock.sendall(message)

    # Look for the response
    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print >>sys.stderr, 'received "%s"' % data

finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```

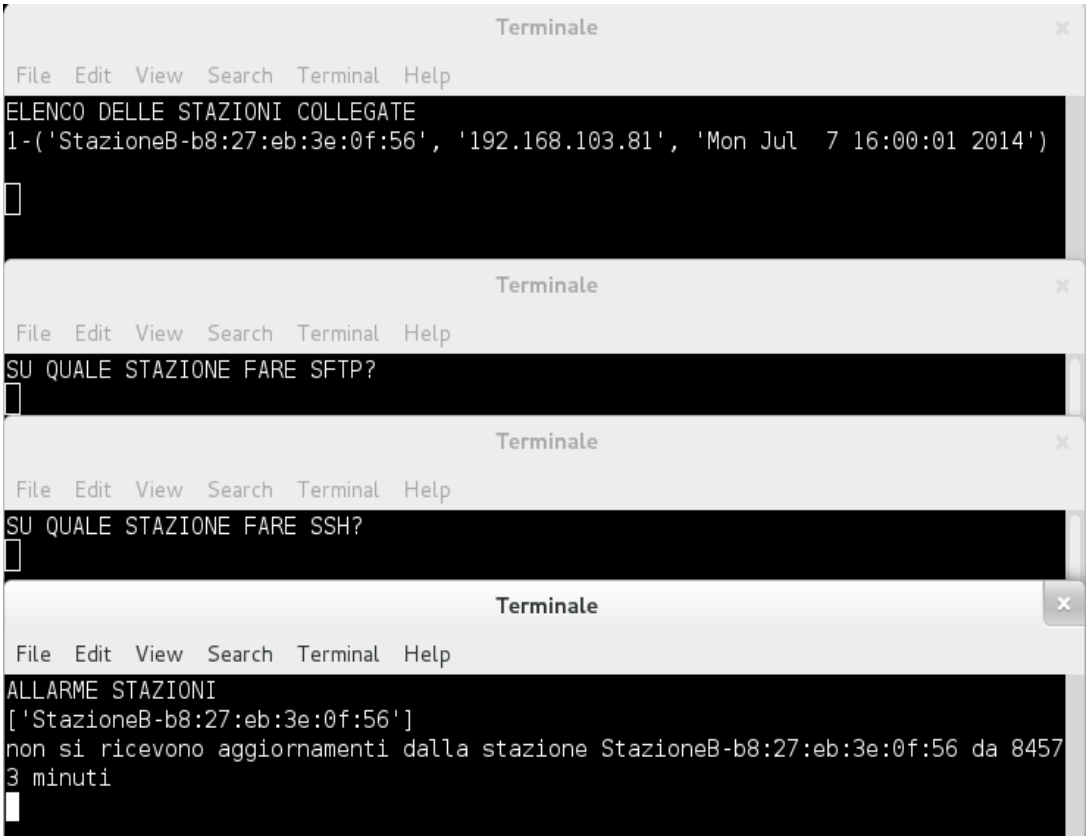

Figura 3.5:screenshot della Stazione Centrale

4 Soluzione wireless

4.1 Introduzione

Abbiamo progettato e realizzato un prototipo di Stazione Monitorante e di Sala Controllo che comunicano attraverso segmenti TCP sopra una rete IP. In uno scenario catastrofico, il mezzo di comunicazione più adeguato è la trasmissione via etere. In questa ottica diventa necessario capire come la trasmissione via etere incida sul sistema Sala Controllo e Stazioni Monitoranti, facendo le seguenti considerazioni:

- **consumo ridotto:** il consumo di corrente da parte delle antenne radio comporta il principale fattore discriminante, volendo realizzare sonde fisicamente isolate da qualsiasi tipo di rete è fondamentale ridurre al minimo il consumo della batteria.
- **area di copertura:** le antenne radio dovranno permettere di creare ponti di un chilometro con un ampiezza di banda utile.
- **economicità:** avendo come obiettivo la realizzazione di stazioni monitoranti con componendi da scaffale,non possiamo considerare come soluzione antenne radio con costo unitario superiore a qualche decina di euro. La singola antenna non può costare quanto il resto della stazione.
- **semplicità:** tra alternative di simili caratteristiche tecniche e costo, opteremo per il protocollo più semplice da implementare.

In particolare, è importare notare che il consumo della batteria dipende dalla trasmissione radio, pertanto optare per antenne a basso consumo energetico significa ridurre l'area di copertura e l'ampiezza di banda. Diventa fondamentale individuare una soluzione efficace, che risulti un giusto compromesso tra i due fattori descritti.

4.2 Moduli radio

Come soluzione, abbiamo pensato di testare i moduli radio, in Figura 4.1, già citati nel capitolo 2. I motivi per i quali ha ritenuto fosse questa la soluzione migliore alle nostre necessità sono:

- costano approssimativamente 25€ l'una
- presentano un'interfaccia seriale, molto semplice da configurare e utilizzare
- in campo aperto il segnale raggiunge un chilometro di distanza
- consumo energetico ridotto
- il modulo è dotato di un micro-controllore che aggiunge ai dati trasmessi un codice CRC per il controllo dell'errore

Come retro della medaglia l'uso di tale tecnologia comporta i seguenti svantaggi:

- il protocollo di comunicazione utilizzato dai moduli è proprietario e quindi opaco
- permette di creare canali di comunicazione a 9600 baud (circa 1,5 kbit/s)

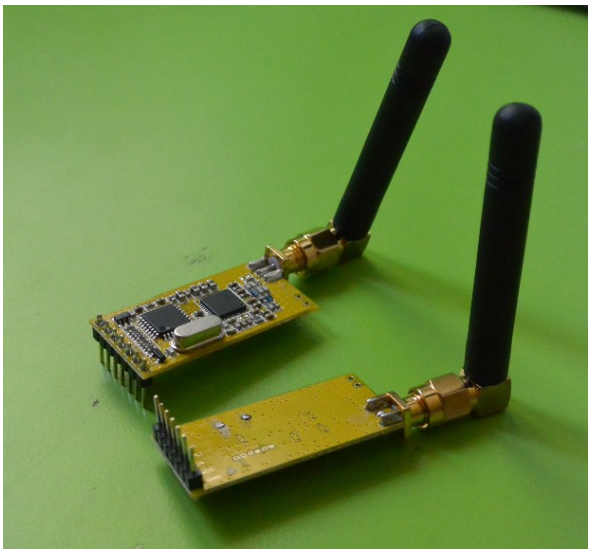

Figura 4.1:moduli radio

4.3 Configurazione della porta seriale

Il Raspberry Pi è dotato di un'interfaccia seriale RS-232. RS-232 è uno standard costituito da una serie di protocolli elettrici ed informatici che rendono possibile lo scambio di informazioni a bassa velocità tra dispositivi digitali.
Il modulo seriale di default è configurato per due scopi:

- trasmettere i messaggi di boot

- permettere il log-in

Per ripulire la comunicazione seriale da tutti i messaggi di output, editare il file /boot/cmdline.txt rimuovendo tutti i riferimenti a: “/dev/ttyAMA0” (nome dell'interfaccia seriale).

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootfstype=ext4 elevator=deadline rootwait
```

La configurazione della porta seriale per il log in si trova nella inittab. Come per il file precedente bisogna rimuovere i riferimenti all'interfaccia /ttyAMA0.

L'antenna va impostata a 9600 baud 8N1 che significa 9600 bit/s, 8 bit per carattere, nessun bit di parità, 1 stop bit.

4.3.1 Utilizzo della porta seriale

Il primo test effettuato è stato fatto lanciando dal terminale della prima macchina il comando:

```
cat /dev/ttyAMA0
```

mentre dal terminale della seconda:

```
echo “Prova di comunicazione 1” > /dev/ttyAMA0
```

È possibile creare utilizzando Python un programma analogo utilizzando l'apposita libreria “serial”[28] per le comunicazioni seriali.

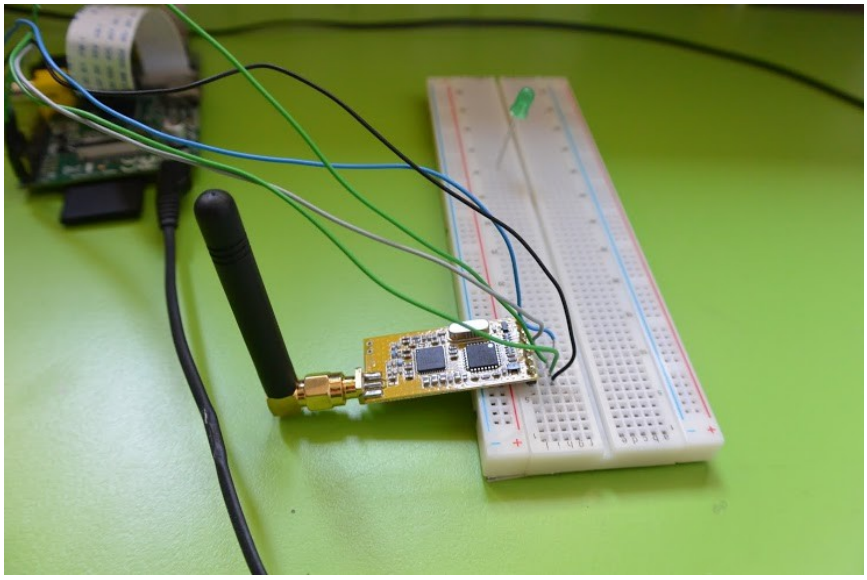

Figura 4.2:Antenna cablata

4.4 Protocollo di Collegamento

Per utilizzare una porta seriale come interfaccia di collegamento abbiamo utilizzato il protocollo SLIP[29] (Serial Line Internet

Protocol). Tale protocollo permette di incapsulare i segmenti IP per spedirli attraverso un collegamento seriale. Il vantaggio di questa soluzione consiste nel poter utilizzare la porta seriale come una qualsiasi interfaccia di rete es: Ethernet o WiFi.

Utilizzando su questo collegamento applicazioni come ssh o scp abbiamo notato una certa lentezza nelle comunicazioni mentre ping non manifesta ritardi.
Il protocollo di trasmissione utilizzato dalle antenne è opaco per cui non è possibile entrare nel dettaglio, tuttavia abbiamo fatto empiricamente le seguenti considerazioni.
TCP è un protocollo affidabile, per garantire questo servizio, oltre ai segmenti contenenti dati, vengono spediti sullo stesso canale segmenti di controllo. Aggiungendo ai pacchetti trasmessi, tutti i pacchetti spediti in senso inverso, il buffer dell'antenna si satura rapidamente. Ping, spedendo pacchetti di 64 bytes, riesce a completare la trasmissione prima che il buffer possa saturarsi. La nostra ipotesi è che le antenne non supportino una comunicazione full-duplex.

5 Conclusione dell'esperienza

L'obiettivo di questa tesi è stato capire se è possibile monitorare fenomeni fisici ambientali utilizzando stazioni a basso costo. Con questa esperienza abbiamo dimostrato che tali stazioni sono realizzabili e abbiamo creato un prototipo utilizzando RaspberryPi. Il prototipo da noi progettato è stato prodotto con componenti il cui costo complessivo è inferiore a 100€.

La possibilità di avere un computer GNU/Linux nelle stazioni monitoranti è resa possibile dalla recente diffusione di CPU ARM a 32 bit per le quali si stanno scrivendo adattamenti di distribuzioni ampiamente utilizzate, ne è un esempio Rasbian (da noi utilizzata) nata da Debian.

Le principali problematiche della stazione emerse e non risolte durante lo sviluppo, quindi da risolvere in futuro, riguardano il miglioramento dei consumi e le comunicazioni wireless.
Per ciò che riguarda i consumi, bisognerà condurre uno studio approfondito per mettere a confronto Raspberry Pi con i tanti altri Computer on a Chip che sono stati commercializzati durante il corso delle nostre sperimentazioni. Tra le tante, vorremmo evidenziare la board A20 Olinuxino prodotta da Olimex (Figura 5.1) poiché consultando la scheda tecnica abbiamo notato le seguenti caratteristiche:

- processore dual core Cortex-A7, il processore di Raspberry è dotato di un singolo core
- 1 GB DDR3 di RAM, Raspberry monta 512MB di Ram
- connettore SATA e connettore per batteria incorporati sulla board, assenti su Raspberry
- ampia gamma di accessori specifici (monitor, batterie, connettori), la Raspberry Pi foundation non produce tali accessori
- basandoci sul datasheet di OlinuxinoA20, rispetto a Raspberry Pi avrebbe un consumo inferiore (da verificare empiricamente)

Per un passo ulteriore nell'ottimizzazione del consumo, bisognerà condurre uno studio sul sistema operativo per valutare se Raspbian,

da noi utilizzato, sia la soluzione ottimale o se esistono alternative più efficienti per lo scopo da noi perseguito. Sarà inoltre necessario analizzare a fondo il sistema operativo scelto, per selezionare, se esistono, demoni e i processi in background inutili alla stazione e bloccarne l'attivazione.

In merito alle soluzioni radio, bisognerà ricercare un dispositivo analogo a quello da noi utilizzato, ma con caratteristiche tecniche idonee ad incapsulare lo stack TCP-IP

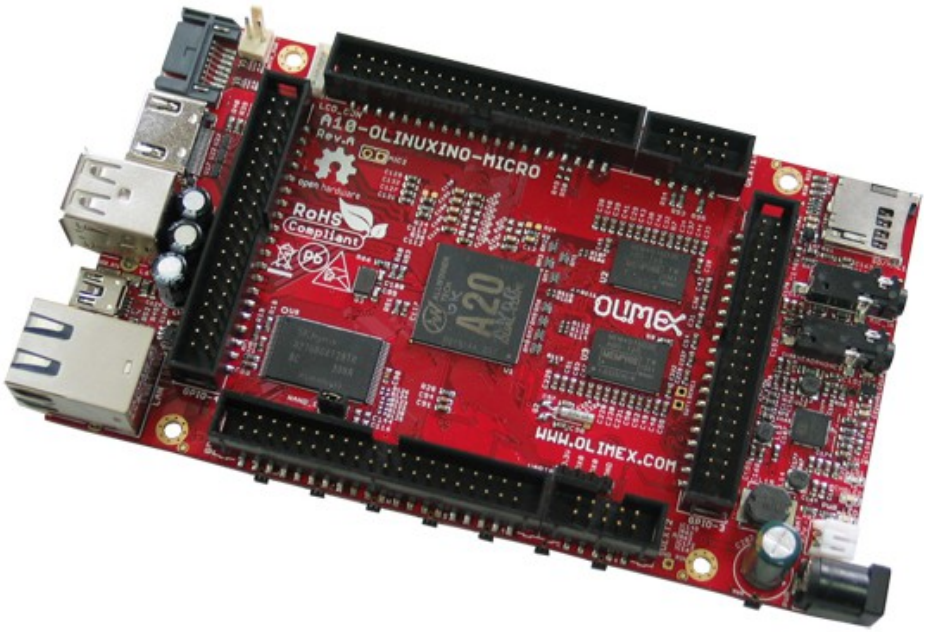

Figura 5.1:Olinuxino A20

INDICE DELLE FIGURE

BIBLIOGRAFIA

[1] INGV-Quaderno di Geofisica 97 anno 2012, pp 10-20
[2] http://www.raspberrypi.org/
[3] http://xbmc.org/
[4] Eben Upton, Gareth Halfacree, Raspberry Pi la guida completa, Hoepli, 2013, p 100
[5] http://www.adafruit.com/
[6] http://httpd.apache.org/
[7] http://www.lighttpd.net/
[8] Linux Pro nr 140 aprile 2014, pp 8-9
[9] Michael Winkoff, Lin Padgham, Developing Intelligent Agent System:A Pratical Guide, 2004, Whiley, p 13
[10] http://commons.wikimedia.org/wiki/File:IntelligentAgent-SimpleReflex.png#mediaviewer/File:IntelligentAgent-SimpleReflex.png
[11] Charles E. Spurgeon, Ethernet: The Definitive Guide, O'Reilly, p 128, 2014
[12] Andreas Molisch, Wireless Communications, Wiley, 2012, cap 24-28-29
[13] Michael Margolis, Arduino Coockbook, O'Reilly, 2013, p 181
[14] Michael Margolis, Arduino Coockbook, O'Reilly, 2013, pp 202-205
[15] www.arduino.cc
[16] www.arduino.cc/en/main/software
[17] Raspberry Pi Camera Module Documentation, http://www.raspberrypi.org/wp-content/uploads/2013/07/RaspiCam-Documentation.pdf
[18] https://www.python.org/
[19] Mark Lutz, Learning Python, O'Reilly, 4th Edition, 2011, pp 247-250
[20] Mark Lutz, Programming Python, O'Reilly, 4th Edition, 2010, pp 189-199
[21] Kurose, Ross, Reti di calcolatori e Internet, Pearson, 4th Edizione, 2008, pp 215-240
[22] Kurose, Ross, Reti di calcolatori e Internet, Pearson, 4th Edizione, 2008, pp 178-180
[23] Simone Piccardi, Amministrare GNU/Linux, Trulite srl, 3° Edizione, pp 560561
[24] Simone Piccardi, Amministrare GNU/Linux, Trulite srl, 3° Edizione, pp 534-537
[25] Adattazione dell'esempio tratto da: Mark Lutz, Programming Python, O'Reilly, 4th Edition, 2010, pp 818-819
[26] Man di clusteredssh
[27] Simone Piccardi, Amministrare GNU/Linux, Trulite srl, 3° Edizione, p 178
[28] http://pyserial.sourceforge.net/
[29] Toni Bautts, Terry Dawson, Linux Network Administrator's Guide, O'Reilly, 3Th Edition, 2005, p 8