ISSN 2039-7941
Anno 2017_Numero 367

rapporti tecnici INGV

Sistema di DNS dinamico utilizzante la messaggistica SMS

367

Istituto Nazionale di Geofisica e Vulcanologia

Segreteria di Redazione
Francesca Di Stefano - Referente
Rossella Celi
Tel. +39 06 51860068
redazionecen@ingv.it

in collaborazione con:
Barbara Angioni (RM1)

REGISTRAZIONE AL TRIBUNALE DI ROMA N.173 | 2014, 23 LUGLIO

Rappresentante legale: Carlo DOGLIONI
Sede: Via di Vigna Murata, 605 | Roma

ISSN 2039-7941
Anno 2017_Numero 367

rapporti tecnici

INGV

SISTEMA DI DNS DINAMICO UTILIZZANTE LA MESSAGGISTICA SMS

Antonino Sicali, Alfio Amantia, Pasqualino Cappuccio

INGV (Istituto Nazionale di Geofisica e Vulcanologia, Sezione di Catania – Osservatorio Etneo)

367

Come citare: Sicali A., Amantia A., Cappuccio P., (2017). Sistema di DNS dinamico utilizzante la messaggistica SMS. Rapp. Tec. INGV, 367: 1-50.

Indice

Introduzione

L'utilizzo di acquisitori sempre più veloci e frequenze di acquisizione sempre più alte richiedono reti di comunicazione all'avanguardia per il trasferimento dei dati. I sistemi di videosorveglianza producono diverse centinaia di *MB* al giorno che, con una normale linea *GSM* o radio a 9600 bps non si riesce a trasferire. Non sempre si possono gestire infrastrutture di rete studiate ad hoc e proprietarie, molte volte si ha l'esigenza e la necessità di utilizzare *ISP* commerciali. Soprattutto i costi di mantenimento e realizzazione delle infrastrutture di rete orientano verso l'utilizzo di reti commerciali. Le reti mobili di ultima generazione, come *UMTS* (*3G*) e *LTE* (*4G*), utilizzano protocolli di comunicazione tipo Internet e fanno largo uso del *packet switch* per ottimizzare l'occupazione di banda. Rispetto ai modem *GSM*, che erano identificate univocamente da un numero telefonico, i nuovi dispositivi di trasmissione, router/modem, impiegano un indirizzo, che pur essendo pubblico, non è statico, e può cambiare spesso. Questo perché il protocollo *IPv4* [Wikipedia, IPv4] non ha disponibili abbastanza indirizzi per tutti i dispositivi ed allo stesso tempo ci sono molti IP condivisi tra dispositivi non collegati. Per ovviare al continuo mutamento degli indirizzi si ha la necessità di associare l'indirizzo numerico *IP* al nome di dominio testuale utilizzando le tecnologie *DNS* [Wikipedia, Domain Name System]. A differenza dell'*IP* il nome di dominio è assegnato univocamente su Internet da organizzazioni internazionali e non subisce variazioni. Il primo utilizzo dei modem UMTS risale all'installazione dei primi dilatometri sul M.te Etna [Sicali et al., 2013], alla fine del 2011. In seguito all'utilizzo di tali strumenti si è sentita l'esigenza di dare una soluzione al problema della rotazione degli indirizzi *IP*, che diventava ogni giorno un problema sempre più serio.

1. Reti tipo Internet

Quando le reti tipo Internet e il protocollo *TCP/IP* sono stati creati alla fine degli anni sessanta del secolo scorso, era impensabile che si sarebbe raggiunta la saturazione degli indirizzi dovuta all'aumento esponenziale dei dispositivi collegati in rete (stima attuale oltre i 4 miliardi). Le scelte di allora, atte a risparmiare spazio di memoria, come è successo molte volte nella storia dell'informatica, portarono a scegliere un indirizzo di soli 32 bit, più che sufficiente per quei tempi, rivelatosi insufficiente ai giorni nostri. Come accadde per il *millennium bug* [Wikipedia, Millennium Bug], causato dall'indicare l'anno con due sole cifre invece che quattro, anche gli indirizzi a 32 bit (*IPv4*) cominciano a creare problemi, dovuti alla scarsa disponibilità degli stessi. Problemi del genere sono normali non solo nell'informatica ma in tutti gli ambiti in cui vengono utilizzati formati. In passato è successo per i dischi fissi di grandi dimensioni, portando all'evoluzione di *filesystem* sempre più moderni ed evoluti e all'adozione del *LBA* [Wikipedia, Logical Block Addressing] nel software d'indirizzamento a basso livello (firmware). In futuro succederà ancora quando nel 2038 i sistemi *POSIX* non saranno più capaci di misurare il tempo [Wikipedia, Bug dell'anno 2038]. Tutto ciò accade quando qualcosa che sembra insuperabile viene prima o poi raggiunta e surclassata, e non è previsto alcun meccanismo di aggiornamento o espansione. L'IPv4 verrà ben presto totalmente sostituito dal più moderno *IPv6* [Wikipedia, IPv6], già implementato ed operativo nei dispositivi tecnologicamente più moderni, ma in attesa dell'adozione su larga scala l'*IPv4* crea un problema particolarmente spinoso per quei dispositivi che vogliono collegarsi a Internet e funzionare da server. Trovare una soluzione per tenere traccia degli *IP* è necessario per non rendere i dispositivi irreperibili. La soluzione più ovvia e corretta è associare il nome di dominio, univoco su Internet, all'indirizzo IP che invece cambia continuamente e utilizzare il *Domain Name Server* (*DNS*)

1.1 Domain Name Server

I dispositivi collegati alle reti tipo Internet sono riconoscibili attraverso un numero di 32 bit. Un dispositivo su una rete tipo Internet potrà essere raggiunto attraverso il suo indirizzo a 32 bit. Un numero di 32 bit nella notazione esadecimale appare come C0A80001, in decimale diviene 3232235521 e nel formato che tutti ormai largamente utilizziamo, diventa 192.168.0.1. Per comodità ogni indirizzo sulle reti tipo Internet viene indicato proprio da quattro numeri decimali separati da punti. Indubbiamente è più comodo indicare un indirizzo con quattro numeri separati da punti rispetto al numero decimale o esadecimale, ma comunque rimane difficile da memorizzare, soprattutto quando gli indirizzi cominciano a essere molti. Si provi a tenere a mente tutti i numeri della propria rubrica telefonica: non è una cosa semplice. Possiamo facilmente rammentare il numero di casa o del lavoro, ma ricordare un centinaio di numeri, soprattutto quelli meno utilizzati è molto più difficile. Più semplice è ricordare i nomi dei nostri conoscenti e amici, piuttosto che il loro numero di telefono. La rubrica serve proprio a questo, ad associare mnemonicamente una informazione più difficile, magari organizzandola gerarchicamente, attraverso nomi, cognomi e altre informazioni simili, rispetto ad avere un numero di telefono completamente anonimo. Bene, il *DNS*

[Wikipedia, Domain Name System] non è altro che una rubrica telefonica enorme in cui ci sono quattro miliardi di numeri telefonici (indirizzi *IP*) associati ad altrettanti nomi. Il sistema naturalmente per essere efficiente è molto complesso e fa uso di molti server che garantiscono la velocità e la ridondanza durante le ricerche. Attraverso il *DNS* si può ovviare in parte alla mancanza di indirizzi *IP*.

1.2 Mancanza d'indirizzi IPv4

Alla fine degli anni sessanta del secolo scorso, al fine di dare supporto alle reti tipo *Internet*, è stata creata, la *suite di protocolli Internet* [Wikipedia, Suite di Procolli Internet], di cui fanno parte i più noti protocolli *TCP* (*Transmission Control Protocol*) e *IP* (*Internet Protocol*). I dispositivi delle reti tipo *Internet* sono riconosciuti attraverso un indirizzo numerico di 32 bit. Anche se negli anni sessanta quattro miliardi di dispositivi sembravano tantissimi, oramai sono divenuti verosimili e la saturazione è stata raggiunta già nel gennaio del 2011 [Wikipedia, Saturazione di IPv4]. Il problema della saturazione degli indirizzi Internet della classe *IPv4* è stato parzialmente risolto, osservando che la maggior parte dei dispositivi richiedono informazioni, ovvero funzionano da *client*, mentre pochi hanno funzionalità di *server*. Sono state create enormi reti private che comunicano attraverso gateway e router, e molte volte riescono a gestire anche richieste in ingresso verso server privati utilizzando il meccanismo del re-indirizzamento delle porte [Wikipedia, Port Forwarding]. Per coloro che necessitano di indirizzi pubblici, gli *ISP* hanno introdotto l'uso degli indirizzi dinamici allo scopo di mitigare il problema della scarsa disponibilità degli indirizzi. Tutto ciò si basa sul concetto che non tutti i dispositivi sono sempre connessi ad Internet, poiché per loro non è conveniente essere connessi continuamente e usano un contratto a consumo. Per quei dispositivi, che invece hanno la necessità di essere sempre disponibili, vengono usati invece contratti di tipo *"flat"*, la cui tariffa non dipende dal tempo di connessione. La rotazione degli indirizzi, che per un contratto a consumo ha senso, viene, forse per semplicità e omogeneità di progettazione dei sistemi software, adottata dagli *ISP* anche per i contratti orientati alla connessione continua. In alcuni casi addirittura non viene rilasciato nemmeno un indirizzo pubblico poiché non è necessario all'utente. Tali strategie, comunque non inficiano il funzionamento della stragrande maggioranza degli utenti. I problemi iniziano quando si ha l'esigenza di essere raggiunti per esplicare un servizio e funzionare da server.

1.3 Problema degli indirizzi dinamici per i Server

Gli indirizzi dinamici divengono un grosso problema per quei dispositivi che devono essere continuamente rintracciabili ovvero tutti quei dispositivi che devono esplicare un servizio, pubblico o privato che sia. Basti pensare ai sistemi di allarme, di monitoraggio o in generale a tutti quei dispositivi di gestione e manutenzione dislocati sul territorio. Il problema sussiste ogni volta che bisogna fornire un servizio. I sistemi di acquisizione in ambito geofisico, espletando un servizio, hanno i medesimi problemi di rintracciabilità. In alcuni casi la telemetria diviene vitale per il buon funzionamento dei sistemi. Ci sono sistemi privi di archiviazione locale adeguata che, in presenza di problemi alla trasmissione, creano buchi enormi nell'acquisizione [Sicali et al., 2016]. La mancanza di un link telemetrico, in altri casi, potrebbe comportare un rischio per la strumentazione scientifica che richiede un assiduo intervento umano. Un indirizzo dinamico per tale classe di dispositivi non è sufficiente a garantire la rintracciabilità ed è necessario un modello che risolva il problema tenendo traccia degli indirizzi.

2. Dynamic Domain Name System

Il *DNS* originariamente è stato creato per migliorare il modo di riferirsi ai dispositivi. Il modello adottato dal *DNS* inizialmente non prevedeva che i record cambiassero, difatti un server *DNS* fa riferimento a dei *file* testuali ad accesso sequenziale, che poco si adattano ad essere continuamente aggiornati in automatico. Nel file testuale a disposizione del server ogni *IP* è associato univocamente a un nome. Con l'introduzione degli indirizzi dinamici si è sentita l'esigenza di estendere le funzionalità del *DNS* introducendo la possibilità di aggiornare dinamicamente i *record:* i *file* testuali sono stati affiancati da strutture dati ad accesso diretto e da protocolli che permettono il loro mutamento. Il nuovo modello è stato ribattezzato *DDNS* [Wikipedia, Dynamic DNS] e permette di sostituire dinamicamente le associazioni tra *IP* e nome, risolvendo pienamente il problema degli indirizzi dinamici per i server.

2.1 DDNS commerciale

Inizialmente la rete di *strainmeters* del M.te Etna utilizzava un servizio *DDNS* commerciale [Sicali et al., 2013]. Oltre al fastidio di aggiornare l'account utilizzato dal router ogni 3 mesi rendendo il sistema non pienamente automatizzabile, i problemi iniziarono quando il servizio, inizialmente libero, divenne a pagamento. Il router utilizzato non permetteva di usare altri tipi di servizi *DDNS* commerciali o privati e le

stazioni sarebbero ben presto divenute irreperibili. Inizialmente si cercò di sostituire i router, scegliendone uno che supportasse un servizio ancora gratuito, ma non si trovò nulla che coprisse completamente le specifiche necessarie. Inoltre scegliendo un altro router, si risolveva il problema solo in parte, visto che ciascun *router* portava con se un numero limitato di servizi *DDNS*. Prima o poi sarebbe stato necessario ricambiarlo, con una spesa non indifferente. Successivamente, visto la disponibilità di un sistema molto flessibile come *Linux,* installato a bordo delle *Shoebox* [Sicali et al., 2013], si cercò un'alternativa che prevedesse l'uso di altri *DDNS* commerciali. Anche questa soluzione era impraticabile poiché l'affidabilità della trasmissione non poteva dipendere da nessun dispositivo di alto livello. Ciò perché i dispositivi di alto livello possono entrare facilmente in stallo, rendendo il canale telemetrico, da cui dipende il buon funzionamento di tutto il sistema [Sicali et al., 2016], inutilizzabile. Si è osservato che utilizzare un *DDNS* commerciale è come utilizzare una *blackbox*: non si ha mai completamente il controllo [Sicali et al., 2016] e molte volte non si riescono a diagnosticare i malfunzionamenti. Per questo motivo si scartarono entrambe le soluzioni in quanto troppo invasive per il sistema, perché lo rendevano instabile. Bisognava allora trovare una soluzione totalmente indipendente e universale, che andava bene, non solo per tutti i sistemi installati, ma anche per quelli utilizzabili in futuro. L'idea di utilizzare un *DDNS* commerciale fu rimpiazzata bene presto dalla possibilità di progettarne e realizzarne uno non commerciale completamente aperto.

2.2 Sistemi DDNS aperti

L'alternativa ai *DDNS* commerciali era creare un nuovo sistema completamente aperto di *DDNS*. Si sono vagliate diverse possibilità. Inizialmente si è pensato di usare un server *DDNS* standard e/o proprietario, in sostituzione di quello commerciale. Tale soluzione è stata subito scartata a causa della complessità dell'operazione. Inoltre l'aggiornamento dei *record* dovrebbe essere affidato a un dispositivo particolarmente complesso, che non sempre è presente nei sistemi di acquisizione. Inoltre gestire un server esterno capace d'interfacciarsi con le stazioni creava problemi di sicurezza informatica e una mole di lavoro per la manutenzione dello stesso non indifferente. Si è vagliata la possibilità di creare un collegamento *VPN* (Virtual Private Network) tra le stazioni e un server esterno. Anche in questo caso i costi della manutenzione non erano indifferenti e inoltre la tecnologia *VPN*, orientata alla connessione, poteva compromettere l'equilibrio energetico delle stazioni, poiché la trasmissione poteva durare parecchio tempo.

Bisognava studiare un meccanismo che allo stesso tempo, fosse semplice, stabile, poco invasivo, che richiedesse poca manutenzione, funzionasse in automatico, completamente immune ai problemi di sicurezza informatica. Fin dalle prime installazioni si utilizzavano dei messaggi *SMS* che il modem spediva in occasione di un riavvio, ma soprattutto quando riceveva un nuovo indirizzo *IP*. Questi messaggi, che venivano visionati solo saltuariamente, in presenza di problemi conclamati, potevano essere la soluzione al problema della tracciabilità degli indirizzi. L'informazione che arrivava con gli *SMS* viaggiava su un canale protetto e perfettamente isolato da tutta la rete informatica del centro di acquisizione. Non risentiva della presenza di firewall o di altri sistemi di sicurezza impiegati e poteva essere processata da un normale sistema software senza particolari problemi. Inoltre essendo già presenti i dispositivi per la ricezione della messaggistica SMS su rete GSM, l'impiego degli stessi non comportava nessuna operazione di rinnovamento tecnologico e infrastrutturale invasivo. La scelta dunque è stata quella di progettare un sistema ad hoc per estrarre le informazioni necessarie da tali *SMS*. Una volta in possesso dell'informazione si possono successivamente aggiornare, in piena sicurezza, tutti i database che richiedono la conoscenza dell'indirizzo *IP* delle stazioni remote. Negli anni si è visto che la consegna di una *SMS* è molto più sicura e affidabile che inviare pacchetti, anche semplici, via Internet. L'affidabilità di tale strumento ha permesso, in passato, di utilizzarli anche per diagnosticare mal funzionamenti al sistema energetico o di trasmissione, poiché molto affidabili in quanto semplici. Quando i livelli OSI [Wikipedia, Open Systems Interconnection] più alti vanno in tilt, a causa di insufficienti risorse, gli *SMS* continuano a funzionare poiché agiscono ad un livello inferiore. In passato si è utilizzata tale proprietà per diagnosticare e discriminare i malfunzionamenti nelle stazioni remote. Il meccanismo è molto semplice. I modem *GSM* molte volte non riuscivano a mantenere la connessione e addirittura non riuscivano ad aprirla. Bisognava capire se il sito fosse afflitto da semplici problemi di trasmissione o più critici problemi di tipo energetico, al fine di organizzare un adeguato intervento tecnico. Si spediva un *SMS* con ricevuta di ritorno al modem *GSM*. Se l'*SMS* veniva ricevuto vuol dire che la stazione era alimentata. Gli *SMS* oggi vengono usati anche come strumento di comando, per impartire ordini a strumentazione remota, e la loro semplicità li rende quasi infallibili.

Funzione	Messaggio
Accensione/Reset Modem	`Router has been powered up. GSM signal strength: -51 dBm.`
Acquisizione nuovo indirizzo IP	`Router has established PPP connection. IP address: www.xxx.yyy.zzz`

Tabella 1. Esempio di *SMS* ricevuti dal modem *UR5i* costruito dalla *Conel.*

2.3 Utilizzo degli SMS come vettore d'informazione

Per trasferire l'indirizzo *IP* all'utilizzatore si possono utilizzare diversi canali di trasmissione. Ciascun canale, appartiene a un particolare livello *OSI*. Ovviamente più in alto, nel modello *OSI* risiederà il canale, maggiori risorse saranno richieste per mantenere la comunicazione. Entrambe le comunicazioni *TCP/IP* e *SMS* si trovano tra il terzo (*data link*) e il quarto livello (*transport*) *OSI*. In teoria dovrebbero avere le medesime prestazioni, ma nella realtà è più semplice ricevere informazioni da un *SMS* che attraverso un pacchetto *IP*. La situazione peggiora considerando i problemi di sicurezza informatica a cui vanno incontro i servizi sulla rete Internet. Un'eventuale sistema non può solo scambiare semplici pacchetti, ma deve anche prevedere un sistema sicuro affinché tale scambio avvenga. La sicurezza, che nel modello *GSM* è implementato nei livelli 1 e 2, nel *TCP/IP* si dovrebbe implementare nei livelli superiori al 4, comportando una perdita di risorse ed un aumento della complessità di tutto il sistema. Ignorando semplicemente la sicurezza informatica e non implementando nulla si esporrebbe la rete aziendale a problemi che nel migliore dei casi potrebbero essere di tipo *denial of service (DoS)* [Wikipedia, Denial of Service].

Gli *SMS* rispetto al *TCP/IP* permettono una maggiore flessibilità degli interventi di manutenzione, permettendo ai *server* di poter avere problemi senza compromettere il loro servizio. Ciò perché gli *SMS* trasformano il canale da sincrono in asincrono. La trasmissione e la ricezione degli *SMS* possono essere temporalmente distanti. Non è necessario che l'utente sia collegato alla rete mobile nel momento dell'invio. La ricezione degli *SMS* può quindi essere realizzata nel momento più adatto, quando si è capace di farlo, e la continua connessione del canale o disponibilità del server non è critico per il buon funzionamento del sistema. Si consideri che nella realtà, per evitare problemi di stallo dei modem questi vengono resettati in automatico dopo un certo numero di ore. Nell'intervallo tra un reset e l'altro, è difficile che venga cambiato l'indirizzo e non è necessario che il server *DDNS* sia online, può anche essere spento o danneggiarsi. Quando il server, dopo la manutenzione, ritornerà funzionante si potranno recuperare le informazioni che non sono state perse durante il malfunzionamento, poiché memorizzate nel database del gestore telefonico, evitando di aspettare diverse ore fino al prossimo reset. L'uso degli *SMS* permette di rendersi indipendenti da un collegamento alla rete Internet o comunque dalla necessità di avere un *IP* pubblico, ovvero di averne uno fisso. Se necessario il server può avere un indirizzo dinamico pubblico o addirittura privato, senza compromettere le funzionalità. Può far parte semplicemente di una rete locale isolata e protetta. Le proprietà asincrone degli *SMS*, permettono nel caso di problemi al server *DDNS*, di risparmiare energia sulle stazioni remote, poiché l'informazione viene spedita un'unica volta. Utilizzando una connessione *TCP/IP* l'informazione verrà trasferita finché qualcuno, il server *DDNS*, la riceverà. Nel caso di un server *DDNS* danneggiato e di un sistema che spedisce l'informazione dell'*IP* progettato male, la trasmissione potrebbe durare molto e far consumare energia alla stazione remota. L'energia sprecata per mantenere inutilmente la trasmissione potrebbe essere utilizzata invece per l'acquisizione e questo permetterebbe al sistema di superare una crisi energetica in corso [Sicali et al., 2016]. Gli *SMS* lavorando a un livello *OSI* basso rispetto agli applicativi e agli altri sistemi software presenti in un sistema di acquisizione, aumentando la precisione nel riconoscimento dei problemi. Una maggiore precisione permette di discriminare anomalie del canale di trasmissione, estendendo l'insieme dei problemi diagnosticabili. Ciò perché, in presenza di anomalie e malfunzionamenti ai livelli OSI alti, gli *SMS* continueranno a spedire informazioni sullo stato di salute del sistema di acquisizione.

L'invio degli *SMS* è stato affidato al router presente nelle stazioni. L'invio da parte del router separa i problemi di trasmissione da quelli di acquisizione permettendo di descrivere meglio le due classi. In presenza di problemi al sistema centrale, gli *SMS* continueranno a fluire, evidenziando la criticità in corso e distinguendola da una in trasmissione o da un problema di tipo energetico. Incrociando tutte le informazioni che arrivano, non solo attraverso gli *SMS* spediti dal router, ma anche attraverso altri canali, come il *TCP/IP,* dall'intero sistema di acquisizione, è possibile descrivere dettagliatamente lo stato di salute delle stazioni remote. Ciò è possibile soprattutto grazie a una corretta e regolare ripartizione dei ruoli tra i vari dispositivi.

Mantenere separati i segnali diagnostici generati è importante per aiutare ad identificare meglio i malfunzionamenti. Oltre l'acquisizione dell'indirizzo *IP,* il router utilizzato può spedire anche altre importanti informazioni sul funzionamento della linea di trasmissione, come per esempio la qualità del segnale radio.

3. Descrizione del software

Il software si compone da diverse parti, com'è mostrato nelle figure 1 e 2. L'attività principale svolta dal software è l'acquisizione dei messaggi provenienti dai diversi *modem router* dislocati sul territorio. Per fare ciò stabilisce una comunicazione di tipo seriale con un modem GSM localizzato nel centro di controllo e utilizzando comandi *AT Hayes* [Wikipedia, Hayes command set] scambia informazioni con il dispositivo. Acquisisce ogni singolo *SMS* che arrivi processandolo per estrarre informazioni utili. Attualmente vengono utilizzati solo due tipi di *SMS*, il primo viene generato all'accensione del modem e contiene la qualità del segnale radio in *dBm (decibel milliwatt)*. Il secondo contiene l'indirizzo *IP* assegnato dall'*ISP* (Internet Service Provider) al *router* remoto (Tabella 1).

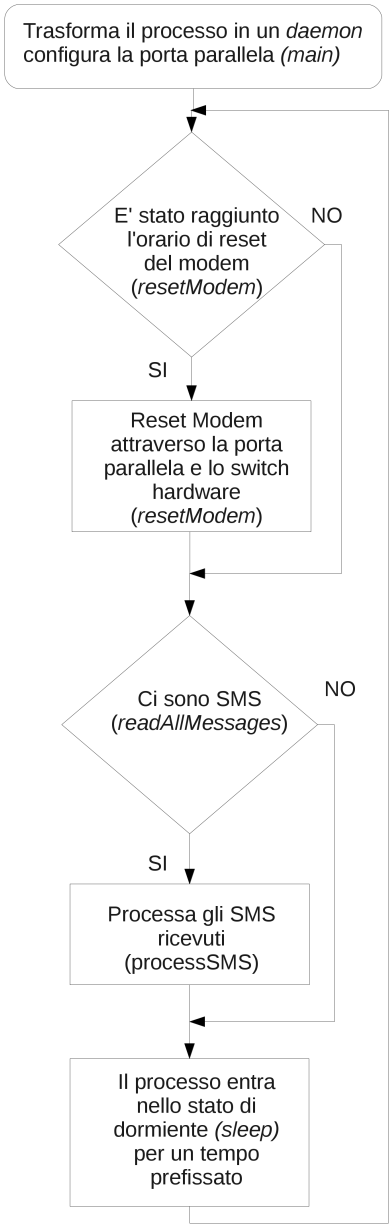

Figura 1. Diagramma di flusso del ciclo di *IDLE* principale, funzione *main.*

3.1 Ciclo di IDLE

Il software può agire in modalità *daemon* o da normale processo. Per funzionare da *daemon* dovrebbe essere definita la macro *ENABLE_DAEMON* durante la compilazione. Si veda in proposito il *Makefile* riportato nell'appendice A5. Da *daemon* il software funziona ininterrottamente come parte integrante del sistema operativo venendo avviato da questo e mantenuto in vita. Per effettuare il *debug*, durante le prime fasi, si è utilizzata la modalità normale che permette di interagire con il software attraverso ambienti di sviluppo *IDE* (Integrated Development Environment) più *user friendly*. Durante l'inizializzazione iniziale il processo, attraverso la funzione *daemon*, viene trasformato in un demone [Wikipedia, Demone], ovvero un programma residente in memoria ed operante in background pronto ad eseguire le funzioni richieste. La funzione *daemon* trasforma il processo in un *daemon* eseguendo un'operazione di *fork cioè creando in memoria una copia di se stesso*. Successivamente sono chiusi lo *standard* di *input*, *output* ed *error*, direzionando questi ultimi due verso il *file* di log definito dalla definizione *LOG_FILE*. Infine viene creato un file, richiesto dallo *script* di avvio (*init.d*), che contiene il *PID* (*Process IDentifier*) caratteristico dei sistemi *unix-like*.

Il software possiede un ciclo di *IDLE* principale all'interno della funzione *main*. Il ciclo di *IDLE* si occupa di eseguire tre operazioni fondamentali: il reset del modem, l'analisi degli *SMS* e l'ingresso nello stato di dormiente (*sleep*), necessario per non consumare inutilmente cicli di *CPU*. Il reset del modem è necessario per evitare, come è già successo in diverse occasioni e applicazioni, che il dispositivo entri in uno stato di stallo, e blocchi il normale funzionamento del sistema. Il reset del modem avviene allo scoccare di ogni ora. Per resettate il modem, il software si avvale di un interruttore elettronico riportato nella figura 5, e comandato attraverso l'*LSB* (*Least Significant Bit*) della porta parallela. Ultimata la procedura di reset del modem, viene acquisita la proprietà della porta seriale (apertura della porta) creando un descrittore di file apposito e configurandola opportunamente. I parametri di configurazione sono quelli classici ovvero 9600 baud, lunghezza parola 8 bit, nessuna parità e 1 bit di stop. Non viene utilizzato nessun segnale di *handshake*.

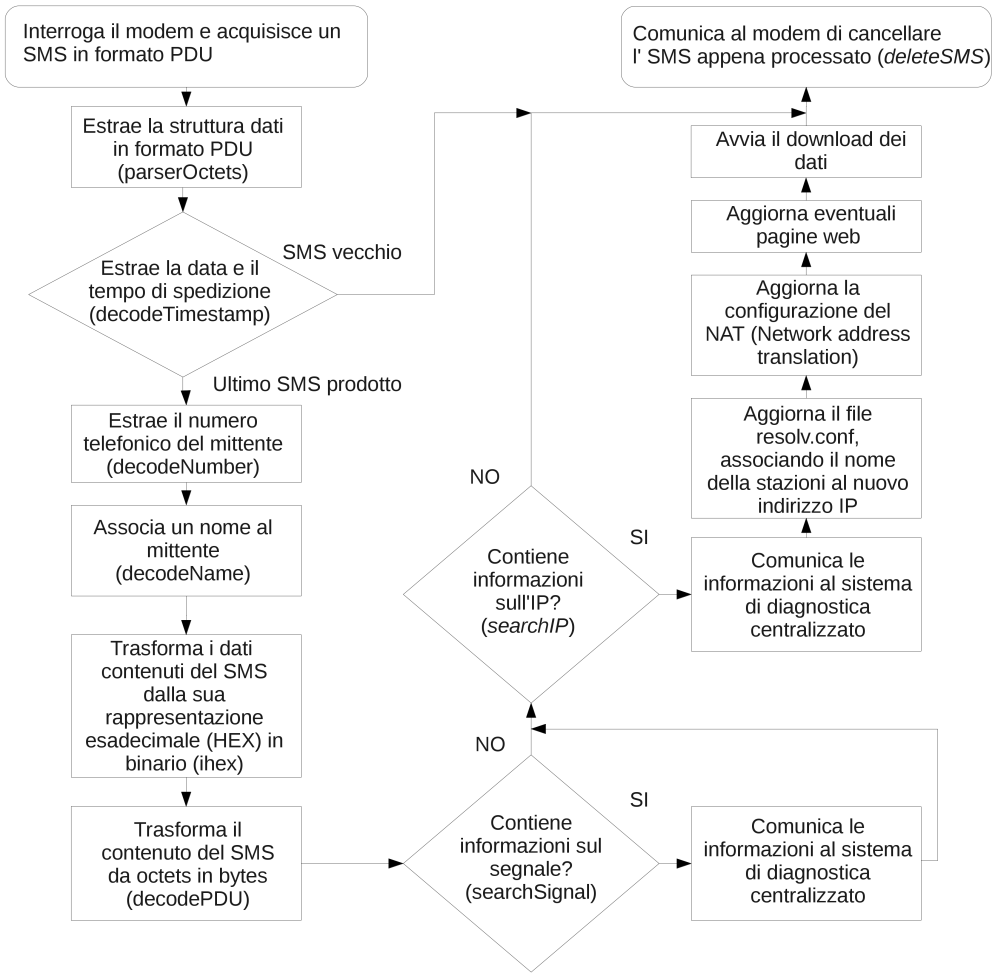

Figura 2. Diagramma di flusso della funzione *processSMS*.

3.2 Processing degli SMS

La funzione *readAllMessages* contiene al suo interno un ciclo che legge dal modem i messaggi memorizzati, sia nella memoria interna che in quella della carta *SIM* [Wikipedia, Subscriber Identity Module]. Ciascun messaggio viene letto e processato attraverso la funzione *processSMS* il suo diagramma di flusso è riportato nella figura 2. Il diagramma sembra molto esteso ma in effetti la funzione esegue linearmente una serie di passi, uno successivo all'altro. Esistono solo due porzioni di codice condizionale che non ramificano comunque il flusso, mantenendolo unico. La funzione inizia con una richiesta di lettura diretta al modem attraverso l'uso dei comandi *AT Hayes* [Wikipedia, *Hayes command set*]. Inviando al modem la stringa *AT+CMGR*, questo replica con un messaggio contenente l'*SMS* in formato *PDU*. Per spedire e ricevere *SMS* attraverso un modem *GSM* si possono usare due differenti formati: testuale e *PDU* (*Protocol Data Unit*). Per un utente normale che deve leggere e spedire messaggi è forse più consigliata la modalità testuale poiché gli *SMS* vengono scambiati in un linguaggio naturale. Tale modalità si attiva utilizzando il comando *AT Hayes AT+CMGF=1*. Il formato *PDU* è da preferire rispetto al testo poiché più simile al linguaggio dei calcolatori. Essendo una semplice struttura dati (array) è più facile e meno ambigua da trattare con un software automatico. Il modem nella modalità *PDU* spedisce e riceve *SMS* nel formato illustrato nella tabella 2.

Ricevuto il messaggio dal modem viene separata la parte propria degli *SMS* dalla rimanente porzione non necessaria. Questa operazione viene eseguita dalla funzione *parserOctets* che ritorna in un buffer una successione di lettere e numeri che riproducono la rappresentazione esadecimale del messaggio, molto simile al formato *Intel HEX* [Wikipedia, Intel HEX]. Solitamente tale formato viene utilizzato per il trasferimento di dati binari attraverso canali testuali, che potrebbero scambiare alcuni caratteri per codici di controllo. Rimossa la parte inutile dal messaggio si può procedere a estrarre la data e il tempo dell'*SMS*, quello che viene chiamato comunemente *timestamp*. Il *timestamp* è stato usato per discriminare i messaggi vecchi da quelli nuovi e non prendere per buono un *IP* oramai vecchio. Succedeva, specialmente durante le prime prove, che arrivassero molti messaggi contemporaneamente e che non veniva rispettato l'ordine di spedizione. Ciò è normale per la messaggistica *SMS* e tutti coloro che hanno usato tale sistema si sono trovati nella situazione di leggere i messaggi in ordine sparso o addirittura dopo molte ore o giorni. Ciò dipende principalmente dal fatto che il dispositivo ricevente potrebbe essere indisponibile durante la trasmissione del messaggio e/o il successivo ordine di consegna non è conforme al tempo di spedizione. Attraverso il *timestamp* è possibile riordinare gli *SMS* e cestinare quelli non più validi. Dal *SMS* viene successivamente estratto il numero telefonico del mittente, attraverso la funzione *decodeNumber*, contenuto nel campo *TP-OA* della struttura dati *PDU* (Tabella 2). Tale numero viene associato dalla funzione *decodeName* a un nome identificativo della stazione di acquisizione. Attraverso tale nome tutti i sistemi che intervengono al funzionamento della catena di acquisizione si riferiscono alla stazione di acquisizione remota. A ogni nome sono associate una serie di caratteristiche di cui fa parte anche l'indirizzo *IP*, si veda in proposito la struttura dati *STATION_FIELD* dichiarata nell'*header file* appendice A1. Identificata la stazione di acquisizione che ha generato il messaggio, si può procedere all'estrazione d'informazioni dei dati contenuti nel *SMS*. Inizialmente vengono convertiti i dati del campo *TP-UD* dal formato *HEX* in binario. Successivamente vengono convertiti gli octets contenuti nel buffer dati in *bytes* per essere processati come stringhe di caratteri dalle normali funzioni di libreria del linguaggio *C*. Il messaggio testuale ottenuto viene analizzato dalle due funzioni *searchSignal* e *searchIP* alla ricerca della qualità del segnale presente nel sito remoto e del nuovo indirizzo *IP*. Le informazioni ottenute vengono inviate al centro di diagnostica per l'analisi dello stato di salute (*heath*). L'indirizzo *IP* viene fornito a tutti i sistemi che ne necessitano per funzionare. L'*SMS* appena processato può infine essere cancellato dalla memoria del modem o della *SIM* attraverso la funzione *deleteSMS* che utilizza il comato *AT Hayes AT+CMGD*.

Nome campo	Nome esteso	Lunghezza	Descrizione
TP-MTI	TP-Message-Type-Indicator	2 bit	Tipo del messaggio
TP-MMS	TP-More-Messages-to-Send	1 bit	Indica se ci sono altri messaggi da spedire
TP-RP	TP-Reply-Path	1 bit	Indica se esistono *Reply Path*
TP-UDHI	TP-User-Data-Header-Indicator	1 bit	Indica se il campo TP-UD contiene un *Header* (opzionale)
TP-SRI	TP-Status-Report-Indication	1 bit	Indica se il dispositivo indicato dalla sigla SME (Short Message Entity) ha richiesto un rapporto sullo stato (opzionale)
TP-OA	TP-Originating-Address	2-12 octets	Indirizzo dell'SME che ha originato il messaggio (numero di telefono)
TP-PID	TP-Protocol-Identifier	octet	Identifica il protocollo del livello
TP-DCS	TP-Data-Coding-Scheme	octet	Indica la codifica del campo TP-UD
TP-SCTS	TP-Service-Centre-Time-Stamp	7 octets	Indica quando il centro servizi ha ricevuto il messaggio
TP-UDL	TP-User-Data-Length	octet	Indica la lunghezza del campo TP-UD
TP-UD	TP-User-Data	octets	Dati (opzionale), il numero di octets è indicato nel campo TP-UDL

Tabella 2. Composizione tipica di un SMS in formato PDU.

3.3 Octets e Caratteri negli SMS

Gli *SMS* fanno uso degli *octets* per misurare la quantità di dati, come mostrato nella tabella 2 [ETSI TC-SMG, 1996]. Nella storia dell'informatica le parole *byte* e *word* hanno assunto sempre significati differenti. Negli anni si è associata erroneamente la lunghezza di otto bit alla parola *byte*. Invece è l'*octet [Wikipedia, Octet]* a essere lungo sempre otto bit, il *byte* può assumere diverse lunghezze [Wikipedia, Byte]. I dati contenuti nel campo *TP-UD* sono *octets* che rappresentano caratteri testuali la cui lunghezza fisica reale è di 7 bit. Per ottimizzare l'impiego della banda (*bandwidth*) il *buffer* in ingresso è stato compresso, risparmiando un *octet* (*byte* di 8 bit) ogni otto caratteri. Per ogni *SMS* spedito si risparmiano 20 *octets*, quindi il messaggio può essere lungo 160 caratteri e occupare lo spazio di 140. I 7 *bit* dei caratteri vengono affiancati a formare una lunga catena (Figura 3c) e successivamente divisi in *octets* (Figura 3b). La funzione che si occupa di convertire (decomprimere) gli *octets* è *decodePDU*. Lavorando direttamente sui *bit* il suo funzionamento può sembrare oscuro e difficile da capire, anche se nella realtà è molto semplice e lineare. Il diagramma di flusso della funzione è riportato nella figura 4.

Gli *octets* vengono letti da un buffer organizzato secondo l'ordine *little indian* [Wikipedia, Ordine dei byte], in cui i byte meno significativi sono posizionati prima. Le operazioni sui bit eseguite nella funzione *decodePDU* potrebbero sembrare strane. Considerando l'organizzazione dei bit e degli octet (Figura 3) tutto diventa più chiaro. In sostanza ogni carattere è scomposto in due porzioni con proporzione crescente e memorizzato in due *octets* adiacenti, per essere ricomposto è necessario spostare alcuni bit: quelli alti, a destra; altri, quelli bassi, a sinistra. Infine gli *octets* nel *buffer* di uscita sono ordinati dal meno significativo al più significativo, come mostrato nella figura 3b. I *bits* meno significativi all'interno di ciascun byte sono invece allineati a destra. I byte adiacenti posseggono una porzione in comune di bit che cresce in modo lineare. Il primo e il secondo *octet* avranno in comune un bit, il bit zero del secondo carattere. Il secondo e il terzo avranno due bit, il terzo e il quarto 3 bit, e così via. Per convertire gli *octets* in caratteri bisogna solo tenere traccia della porzione di bit che ogni *i-esimo octet* del *buffer* contiene. Ciò viene fatto attraverso la variabile locale *bits*. Esiste anche un'altra variabile locale, *mask*, che viene utilizzata per la mascheratura degli *octets*, ovvero l'operazione che mette in risalto attraverso una operazione logica *AND* alcuni bit e riportando a zero gli altri. Le due porzioni individuate da *mask* e dalla sua negazione vengono fuse insieme dopo essere state opportunamente traslate (*shift*) a destra e sinistra. Facciamo un esempio esplicativo e supponiamo che si voglia estrarre il carattere blu della figura 3. Tale carattere è diviso tra il secondo e il terzo *octet*. La variabile bits conterrà il valore 1 poiché le proporzioni sono 2 e 5, mentre lo spazio di bits varia tra

0 e 6, e mappa i valori tra 1 e 7. La variabile *mask* conterrà il valore binario 00011111 ovvero la rappresentazione dei *bit* blu nel terzo *octet* della figura 3b. Applicando la maschera al terzo *octet,* spostando tale valore verso sinistra di *bits*+1 e combinandolo al valore frutto della mascheratura al ciclo precedente del secondo *octet*, spostato a destra di 7-bits, attraverso un'operazione di logica *OR*, si ottiene il carattere richiesto. L'operazione apparentemente complessa, ha fatto risparmiare 4 bit (caselle grigie nella figura 3c), e nel conteggio finale permette di trasferire 160 caratteri nello spazio di 140.

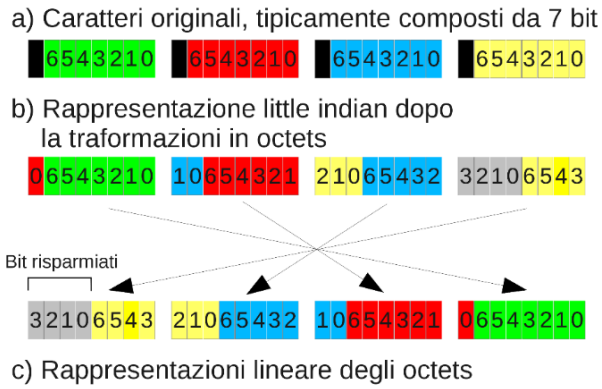

Figura 3. Schema rappresentativo della trasformazione dei caratteri in octets.

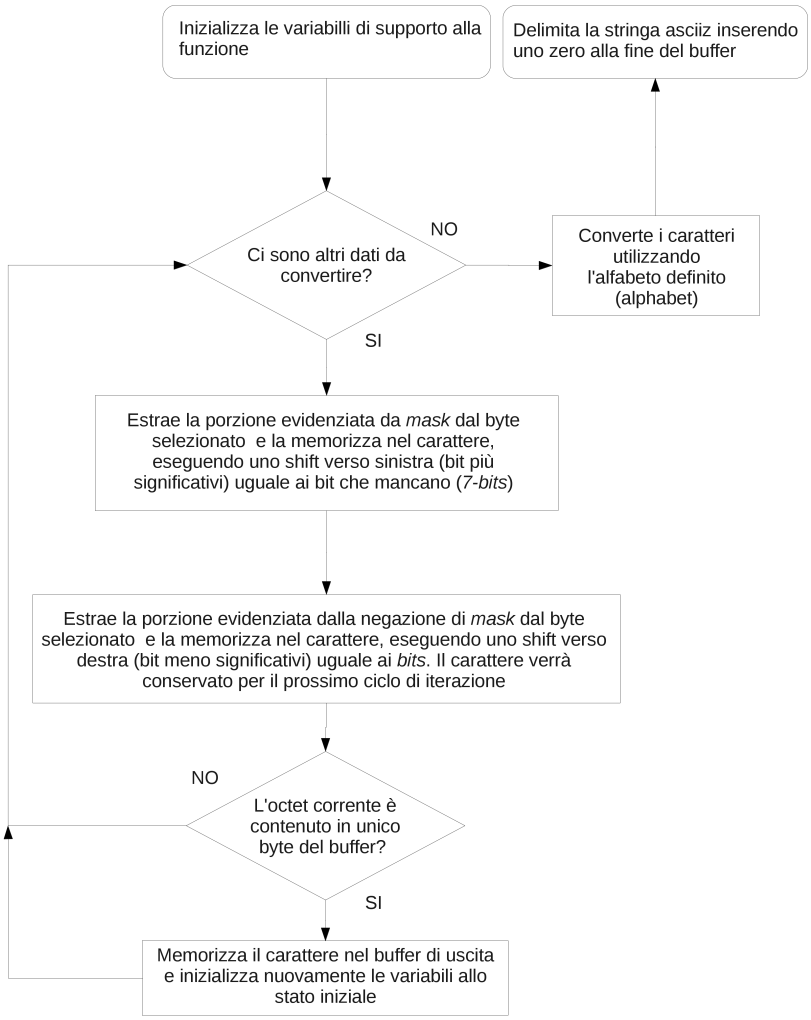

Figura 4. Diagramma di flusso della funzione decodePDU.

3.4 Configurazione del software

Nell'*header file* si può trovare una sezione *Configurations* che contiene una serie di definizioni che permettono di attivare o meno porzioni di codice secondo le diverse esigenze. In base alla complessità del sistema e ai servizi offerti possono essere combinate insieme tali definizioni.

La definizione *ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS* permette di abilitare il trasferimento delle informazioni al software centralizzato di diagnostica, che provvederà a incrociarle per ricavarne lo stato di salute (*health*) della stazione remota di acquisizione.

Le definizioni *RDAT_USER* e *RDAT_PASSWORD* indicano le credenziali di accesso al sistema di diagnostica centrale.

La definizione *IDLE_DELAY_TIME* indica l'intervallo intercorrente tra un invio e l'altro dei messaggi di diagnostica di tipo *IDLE* diretti al sistema di diagnostica centralizzata. Il sistema di diagnostica riceve tali messaggi e verifica che il sistema *DDNS* funzioni.

La definizione *ENABLE_WEB_PAGES* permette di aggiornare alcune informazioni presenti su alcuni siti web di accesso remoto. In particolare vengono generati dei link e rilasciato l'IP attraverso il protocollo *http* che uno script esterno alla rete locale può utilizzare per accedere al sito remoto di acquisizione.

La definizione *ENABLE_NAT* viene utilizzata per abilitare la funzionalità che permette a un sistema di acquisizione di poter essere raggiunto attraverso un indirizzo numerico fisso. Ci sono software di scaricamento dati così elementari che non possono accedere alla stazione attraverso un nome. Abilitando questa funzionalità si dà l'opportunità anche a questi di potersi collegare ai siti remoti.

La definizione *ENABLE_LOCAL_DNS* permette di aggiornare i record dei nomi locali. Questa funzionalità può essere completamente rimpiazzata da *ENABLE_NSUPDATE*, se i servizi che fanno uso degli *IP* non sono solo locali.

La definizione *ENABLE_NSUPDATE* permette di abilitare la funzionalità che modifica i record di un server *DDNS* locale, permettendo a qualsiasi sistema possa utilizzare il nome di accedere alle risorse e collegarsi ai siti remoti.

La definizione *ENABLE_DOWNLOAD* permette di rendere l'operazione di scaricamento dei dati asincrona, permettendo un'ottimizzazione delle risorse e aumentando le probabilità di connessione ai siti remoti in presenza di canali di trasmissione molto deteriorati. Ciò perché molte volte il canale è disponibile per un periodo di tempo molto limitato e l'interrogazione effettuata ciclicamente potrebbe non essere inclusa in tale intervallo. Diversamente, quando il canale è disponibile viene avviata la procedura per la raccolta dati subito dopo la ricezione degli *SMS*.

La definizione *STRAIN_USER* indica l'utente di accesso ai sistemi distribuiti e di supporto all'acquisizione. Tale utente viene usato per il backup dei dati e per altri servizi simili.

La definizione *SWITCH_PORT* indica l'indirizzo della porta parallela a cui è collegato l'interruttore elettronico per il reset del modem.

La definizione *MODEM_PORT* indica il nome della porta seriale a cui è collegato il modem *GSM*.

La definizione *NSTATIONS* indica al sistema quante stazioni deve gestire. Il nome delle stazioni e il relativo numero *GSM* vengono riportate nelle definizioni *XXX_NAME* e *XXX_NUMBER*.

La definizione *RESET_HOURS_DELAY* indica il numero di ore intercorrente tra un reset e l'altro del modem *GSM* che riceve gli *SMS*.

La definizione *SMS_DELAY_TIME* indica l'intervallo intercorrente tra un'interrogazione e l'altra del modem alla ricerca di nuovi *SMS*.

4. Descrizione dell'hardware

Il software per funzionare ha bisogno di un dispositivo, un modem *GSM*, per ricevere gli *SMS*. I dispositivi di alto livello, come i modem, soprattutto se facenti parte della fascia consumer, possono andare in errore [Sicali et al., 2016]. Per ovviare ai malfunzionamenti e avarie del sistema, ogni ora, viene preventivamente resettato il modem *GSM* attraverso l'utilizzo dell'interruttore elettronico riportato in figura 5. Il software utilizza la porta parallela come interfaccia *GPIO* (*General Purpose Input/Output*) [Wikipedia, General Purpose Input/Output] per controllare l'interruttore hardware. L'interruttore impiega un relè per assicurare il massimo isolamento tra il modem e il computer. L'alimentazione dell'interruttore viene prelevata attraverso una presa *Molex* serie 8981 dall'alimentatore centrale del *PC* su cui è installato il software. Volendo si può alimentare il modem anche attraverso tale connettore, però rinunciando all'isolamento tra i due dispositivi. Generalmente non è importante isolare i due dispositivi, però molte volte in presenza di antenne esterne potrebbe essere necessario per evitare la propagazione delle extra tensioni da fulmine.

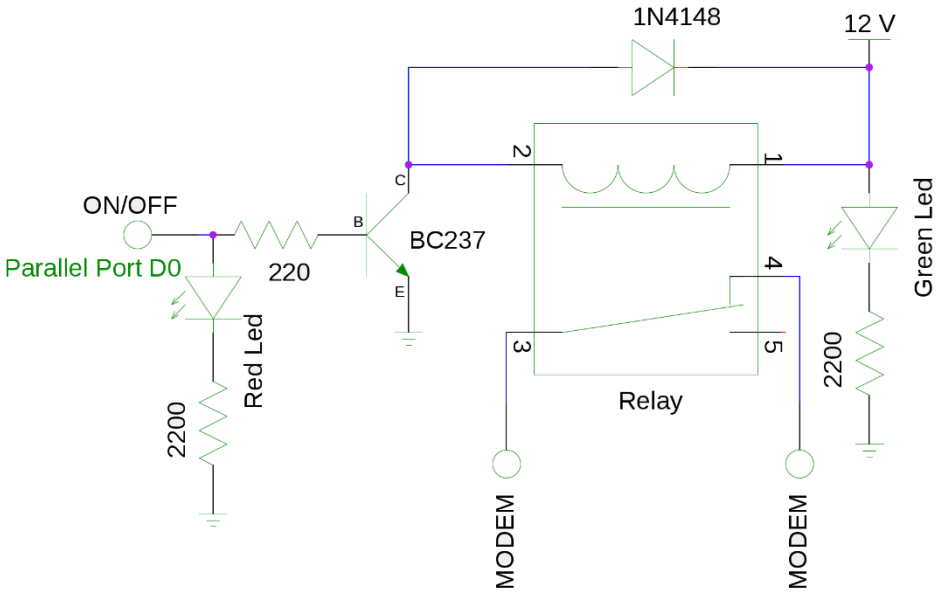

Figura 5. Interruttore elettronico per il reset hardware del modem *GSM*.

5. Funzionalità aggiuntive

Utilizzando un sistema non commerciale, completamente aperto e programmabile si possono implementare dei servizi personalizzati che potrebbero migliorare la stabilità, l'efficienza e automatizzare alcuni processi [Sicali et al., 2016]. Si possono creare sistemi capace di venire in contro a qualsiasi tipo di esigenze. Nell'attuale versione del software sono stati implementati diversi servizi che possono aiutare l'intera catena di acquisizione.

La prima funzionalità permette di trasformare l'interrogazione e il trasferimento dei dati in un'azione di tipo asincrona e priva di ritardi e latenze. Trasformare l'operazione di recupero di dati da sincrona ad asincrona permette di trasmettere anche in presenza di canali di trasmissione molto instabili. Questo perché l'interrogazione della stazione remota avviene sempre in prossimità del periodo di disponibilità del canale di trasmissione. Il canale di trasmissione molte volte può essere limitato nel tempo e potrebbe non coincidere con la richiesta ciclica sincrona dello *scheduler* normalmente usato.

Il secondo servizio, la diagnostica distribuita ed integrata, prevede di inserire segnali a tutti i livelli che permettono di aiutare l'amministratore nel lavoro di diagnostica e intervento. Un sistema di diagnostica che aspira alla perfezione deve creare una maglia così stretta di segnali tale che nessun problema o malfunzionamento può passare inosservato. La diagnostica implementata nel sistema DDNS usato è stata possibile solo perché completamente conosciuta e aperta pur mantenendo alti gradi di sicurezza intrinseca.

La distribuzione degli indirizzi *IP* può essere personalizzata e adattata per ciascun sistema di acquisizione. Non tutti i sistemi possono ricevere un indirizzo o nome che cambia. Sono stati usati sistemi, poco flessibili e che non permettevano di indicare un *domain name* ma solo un indirizzo *IP,* difficile da modificare in automatico attraverso un software. Per tali sistemi si è usato lo script riportato nell'appendice A6. Tale script permette di configurare il *NAT (Network Address Traslation*) di *Linux* per instradare i pacchetti *TCP/IP* e mantenere l'*IP* fisso, come richiesto dal sistema di acquisizione. Il computer su cui risiede il software di gestione del sistema di acquisizione deve solo collegarsi via *VPN* al server *DDNS.* Per tutti gli altri sistemi di acquisizione più evoluti, che possono usare direttamente il nome invece del solo *IP,* è stato utilizzato un server *DNS* dinamico che utilizza l'utility *NSUPDATE* per aggiornare i vari record. Se i servizi sono solamente locali, si possono usare meccanismi più semplici del server *DNS.* Per trasferire all'esterno della rete gli *IP* si è scelto il protocollo http [Wikipedia, Hypertext Transfer Protocol], che permette di trasferire in modo semplice l'informazione a terze parti, evitando di gestire server *DDNS* pubblici e andare in contro a problemi di sicurezza.

6. Installazione e utilizzo

Il software non richiede particolari accorgimenti per funzionare. Nell'appendice A5 è riportato il *Makefile* necessario per compilarlo. Compilando il software attraverso l'utente *root* viene contemporaneamente installato nella corretta *directory*. Il software viene richiamato durante il *boot* attraverso il meccanismo dell'*init.d*. Nell'appendice A3 è riportato lo script usato dall'*init.d*. Tale script potrebbe essere riadattato secondo il sistema utilizzato. Solitamente nella directory */etc/init.d* esiste uno *script di shell* chiamato *skeleton* che rappresenta un modello del tipico script di avvio dei *daemons* attraverso *init.d*. Nel caso in cui si hanno problemi con lo script dell'appendice A3 si dovrebbero solo inserire le informazioni di tale script nel modello *skeleton*. Le informazioni da aggiornare e che si dovrebbero trasferire al nuovo script sono essenzialmente due:

```
DESC="Dynamic DNS through GSM Short Message"
NAME=ddns
```

Infine il software ha bisogno di una porta parallela per resettare il modem e una seriale per le comunicazioni. Bisognerà verificare che tali porte nel software sono state configurate correttamente attraverso le due definizioni *SWITCH_PORT* e *MODEM_PORT* riportate nell'*header file* (Appendice A1).

Conclusioni

Il problema degli indirizzi pubblici dinamici, per i sistemi dislocati sul territorio che eseguono un monitoraggio in continuo e devono essere sempre reperibili, è molto importante ma di non facile soluzione. Ogni sistema che risolve tale problema potrebbe introdurne altri. Alcune soluzioni potrebbero essere più nocive del problema stesso, soprattutto se incidono sulla stabilità e sull'efficienza. Per i sistemi di monitoraggio alimentati a energia solare il sistema del *DDNS* impiegante gli *SMS* si è rivelato ottimo e robusto poiché poco invasivo, soprattutto per quanto concerne l'efficienza energetica e la stabilità. Inoltre, la semplicità del modello adottato permette di aumentare la precisione della diagnostica ed estendere l'insieme dei problemi riconosciuti.

Ringraziamenti

Volevamo ringraziare tutta la Segreteria di Redazione del CEN che si è dimostrata, ancora una volta, molto veloce ed efficiente. Un ringraziamento particolare è dovuto alla dott.ssa Rossella Celi per la cordialità e la disponibilità mostrata, ed al revisore Dr. William Thorossian per l'accuratezza e la precisione adottata durante la revisione.

Bibliografia

ETSI TC-SMG (1995). *Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information*. GSM 03.38.

ETSI TC-SMG (1996). *Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS) Point-to-Point (PP)*. GSM 03.40.

Sicali A., Bonaccorso A., (2013). *Gestione dei dilatometri installati in pozzi profondi all'Etna*. Rapporti Tecnici INGV n°. 258, ISSN 2039-7941.

Sicali A., Amantia A., Cappuccio P., (2016). *Linee guida e criticità nella progettazione di sistemi per l'acquisizione di dati geofisici in prossimità di vulcani attivi.* Rapporti Tecnici INGV n°. 347, ISSN 2039-7941.

Sitografia

Wikipedia, *IPv4*, https://it.wikipedia.org/wiki/IPv4.

Wikipedia, *Domain Name System*, https://it.wikipedia.org/wiki/Domain_Name_System.

Wikipedia, *Millennium Bug*, https://it.wikipedia.org/wiki/Millennium_bug.
Wikipedia, *Bug dell'anno 2038*, https://it.wikipedia.org/wiki/Bug_dell%27anno_2038.
Wikipedia, *Logical Block Addressing*, https://it.wikipedia.org/wiki/Logical_block_addressing.
Wikipedia, *IPv6*, https://it.wikipedia.org/wiki/IPv6.
Wikipedia, *Saturazione di IPv4*, https://it.wikipedia.org/wiki/Saturazione_di_IPv4.
Wikipedia, *Port Forwarding*, https://en.wikipedia.org/wiki/Port_forwarding.
Wikipedia, *Dynamic DNS*, https://it.wikipedia.org/wiki/Dynamic_DNS.
Wikipedia, *Hayes command set*, https://en.wikipedia.org/wiki/Hayes_command_set.
Wikipedia, *General Purpose Input/Output*, https://it.wikipedia.org/wiki/General_Purpose_Input/Output.
Wikipedia, *Subscriber identity module*, https://en.wikipedia.org/wiki/Subscriber_identity_module.
Wikipedia, *Intel HEX*, https://en.wikipedia.org/wiki/Intel_HEX.
Wikipedia, *Octet*, https://en.wikipedia.org/wiki/Octet_(computing).
Wikipedia, *Byte*, https://it.wikipedia.org/wiki/Byte.
Wikipedia, *Demone*, https://it.wikipedia.org/wiki/Demone_(informatica).
Wikipedia, *Denial of Service*, https://it.wikipedia.org/wiki/Denial_of_service.
Wikipedia, *Ordine dei byte*, https://it.wikipedia.org/wiki/Ordine_dei_byte.
Wikipedia, *Open Systems Interconnection*, https://it.wikipedia.org/wiki/Open_Systems_Interconnection.
Wikipedia, *Hypetext Transfer Protocol*, https://it.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

ISSN 2039-7941
Anno 2017_Numero 367

Appendici

Appendice A1. Codice sorgente: ddns.h

```
#ifndef _DDNS_H
#define _DDNS_H

#define LOG_FILE "ddns.run.log"
#define SMS_LOG_FILE "ddns.sms.log"

#ifdef _ENABLE_DAEMON
#define HOME_DIRECTORY "/var/ddns"
#define PID_FILE "/var/run/ddns.pid"
#else
#define HOME_DIRECTORY "/home/strain/Shoebox/ddns"
#define _DISABLE_DELETE
#endif

#define PDU_DATA_ADDRESS          27*2
#define PDU_DATA_ADDRESS_SIZE     PDU_DATA_ADDRESS-2
#define PDU_NUMBER_ADDRESS        11*2
#define PDU_NUMBER_ADDRESS_SIZE   9*2
#define PDU_TIMESTAMP_ADDRESS     19*2
#define PDU_TIMESTAMP_SIZE        7*2

#define OK_MESSAGE "OK"
#define CMGR_MESSAGE "+CMGR:"
#define CMS_ERROR_MESSAGE "+CMS ERROR"
#define ADDRESS_MESSAGE "address"
#define SIGNAL_MESSAGE "strength"
#define CONFIGURATION "at&f\r\n"

#define OCTET_SIZE 7

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                          Configurations                            //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

#define ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
#define ENABLE_WEB_PAGES
#define ENABLE_NAT
#define ENABLE_LOCAL_DNS
#define ENABLE_DOWNLOAD
#define ENABLE_NSUPDATE

//#define _DISABLE_DELETE

#define RDAT_USER "strain"
#define RDAT_PASSWORD "password"

#define STRAIN_USER RDAT_USER

#define RESET_HOURS_DELAY 1

#define SWITCH_PORT 0x378   //LPT0
#define MODEM_PORT  "/dev/ttyS0"

#define NSTATIONS 5

#define DRUV_NAME "druv"
```

```
#define DEGI_NAME "degi"
#define DPDN_NAME "dpdn"
#define DMSC_NAME "dmsc"
#define CNE_NAME  "cne"

#define DRUV_NUMBER "+39"
#define DEGI_NUMBER "+39"
#define DPDN_NUMBER "+39"
#define DMSC_NUMBER "+39"
#define CNE_NUMBER  "+39"

#define SMS_DELAY_TIME  10
#define IDLE_DELAY_TIME 300

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                        Strutture dati                              //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

typedef struct DT {
        int day;
        int month;
        int year;
        int hour;
        int min;
        int sec;
} DT;

typedef struct STATION_FIELD {
        char name[32];
        char number[32];
        DT dt;
        char ip[32];
        char signal[32];
} STATION_FIELD;

int query(int fd,unsigned char *s,unsigned char *tmp);
bool searchField(unsigned char *s,const char *field,char *tmp);

#endif
```

Appendice A2. Codice sorgente: ddns.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <termios.h>
#include <time.h>
#include <fcntl.h>
#include <linux/fs.h>
#include <linux/limits.h>
#include <sys/io.h>
#include <sys/ioctl.h>
#include "ddns.h"

/*
        Indica il file cui verranno scritti i messaggi di logging
*/
FILE *HLOG=stdout;


/*
        Lista delle stazioni da gestire
*/
STATION_FIELD stations[NSTATIONS]={{DRUV_NAME,DRUV_NUMBER,{0,0,0,0,0,0},"",""},
                                   {DEGI_NAME,DEGI_NUMBER,{0,0,0,0,0,0},"",""},
                                   {DPDN_NAME,DPDN_NUMBER,{0,0,0,0,0,0},"",""},
                                   {DMSC_NAME,DMSC_NUMBER,{0,0,0,0,0,0},"",""},
                                   {CNE_NAME ,CNE_NUMBER ,{0,0,0,0,0,0},"",""}};

/*
        Default Alphabet table, ETSI GSM 03.38
*/
unsigned short alphabet[128]={64 ,163,36 ,165 ,232,233,249,236,
                              242,199,10 ,216 ,248,13 ,197,229,
                              916,95 ,934,915 ,923,937,928,936,
                              931,920,926,8364,198,230,223,201,
                              32 ,33 ,34 ,35  ,164,37 ,38 ,39 ,
                              40 ,41 ,42 ,43  ,44 ,45 ,46 ,47 ,
                              48 ,49 ,50 ,51  ,52 ,53 ,54 ,55 ,
                              56 ,57 ,58 ,59  ,60 ,61 ,62 ,63 ,
                              161,65 ,66 ,67  ,68 ,69 ,70 ,71 ,
                              72 ,73 ,74 ,75  ,76 ,77 ,78 ,79 ,
                              80 ,81 ,82 ,83  ,84 ,85 ,86 ,87 ,
                              88 ,89 ,90 ,91  ,92 ,93 ,94 ,95 ,
                              96 ,97 ,98 ,99  ,100,101,102,103,
                              104,105,106,107 ,108,109,110,111,
                              112,113,114,115 ,116,117,118,119,
                              120,121,122,123 ,124,125,126,127};

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//               Funzioni per distribuzione dei nuovi IP              //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Aggionra il database del DDNS locale per permettere a chiunque
```

```
        di ricevere il nuovo indirizzo IP     della stazione
*/
void nsupdate(unsigned char *name,unsigned char *ip)
{
      char cmd[1024];
      char filename[1024];

      sprintf(filename,"%s/nsupdate.cmd",HOME_DIRECTORY);
      FILE *handle=fopen(filename,"wt+");
      fprintf(handle,"server 127.0.0.1\n");
      fprintf(handle,"zone ddns.ct.int.ingv.it\n"),
      fprintf(handle,"update delete %s.ddns.ct.int.ingv.it. A\n",name);
      fprintf(handle,"update add %s.ddns.ct.int.ingv.it. 60 A %s\n",name,ip);
      fprintf(handle,"show\n");
      fprintf(handle,"send\n");
      fprintf(handle,"quit\n");
      fclose(handle);

      sprintf(cmd,"nsupdate -k
      /etc/bind/Kctstrainmeters.ct.int.ingv.it.+157+15493.private %s/nsupdate.cm
      d",HOME_DIRECTORY);
      system(cmd);
}

/*
        Crea una pagina web per il reindirizzamento automatico al sito web
        della stazione
*/
#ifdef ENABLE_WEB_PAGES
void writeRedirect(unsigned char *name,unsigned char *ip)
{
    FILE *handle;
    char filename[256];

    sprintf(filename,"%s/%s/index.html",HOME_DIRECTORY,name);
    handle=fopen(filename,"wt+");

    fprintf(handle,"<html>\n");
    fprintf(handle,"<script language=\"JavaScript\">\n");
    fprintf(handle,"var url=\"http://%s\";\n",ip);
    fprintf(handle,"if (document.images)\n");
    fprintf(handle,"            location.replace(url);\n");
    fprintf(handle,"    else\n");
    fprintf(handle,"            location.href = url;\n");
    fprintf(handle,"</script>\n");
    fprintf(handle,"<a href=\"http://%s\">Wait please...</a>\n",ip);
    fprintf(handle,"</html>\n");
    fclose(handle);
}
#endif

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//       Funzioni per la manipolazione delle strutture temporali      //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Confronta due strutture temporali
*/
```

```
int CmpTime(const DT &a,const DT &b)
{

         if (a.year < b.year) return -1;
         if (a.year > b.year) return 1;
         if (a.month < b.month) return -1;
         if (a.month > b.month) return 1;
         if (a.day < b.day) return -1;
         if (a.day > b.day) return 1;
         if (a.hour < b.hour) return -1;
         if (a.hour > b.hour) return 1;
         if (a.min < b.min) return -1;
         if (a.min > b.min) return 1;
         if (a.sec < b.sec) return -1;
         if (a.sec > b.sec) return 1;

     return 0;
}

/*
        Operatore minore o uguale per le strutture temporali di tipo DT
*/
bool operator<=(const DT &a,const DT&b)
{
        return CmpTime(a,b) <= 0;
}

////////////////////////////////////////////////////////////////////////////
//                                                                        //
//                                                                        //
//           Funzioni per la manipolazione di stringhe                    //
//                                                                        //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

/*
        Elimina i caratteri non importanti come gli spazi, CR, LF alla fine e
        all'inizio della stringa
*/
void trim(char *line)
{
        int j=0,i;

        if (!line[0]) return;

        i=0;
        while(line[i]) {
                                if (line[i] == '\t') line[i]=0x20;
                                i++;
        }

        i=0;
        while(line[i] && (line[i] <= 0x20 || line[i] >= 0x80)) i++;
        if (!line[i]) { line[0]=0; return; }

        while(line[j+i]) {
                line[j]=line[j+i];
                j++;
        }

        line[j]=0;
```

```
        while(line[j] <= 0x20 || line[j] >= 0x80) j--;
        line[j+1]=0;
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                    Funzioni di supporto al debug                   //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Scrive la data e l'orario sullo standard di output
*/
void writeDate(void)
{
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    printf("\n%4.4hd-%2.2hd-%2.2hd %2.2hd:%2.2hd:%2.2hd ",
                tm.tm_year + 1900,tm.tm_mon + 1,tm.tm_mday,
                tm.tm_hour,tm.tm_min,tm.tm_sec);
}

/*
        Scrive la stringa sullo standard di output
*/
void writeLog(char *format,unsigned char *s)
{
    char filename[256];
    sprintf(filename,"%s/%s",HOME_DIRECTORY,SMS_LOG_FILE);
    FILE *handle=fopen(filename,"at+");
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);
    fprintf(handle,"%4.4hd-%2.2hd-%2.2hd %2.2hd:%2.2hd:%2.2hd ",
                tm.tm_year + 1900,tm.tm_mon + 1,tm.tm_mday,
                tm.tm_hour,tm.tm_min,tm.tm_sec);
    fprintf(handle,format,s);
    fprintf(handle,"\n");
    fclose(handle);
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                  Funzioni di gestione del modem                    //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

#ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS

/*
        Ricerca il contenuto di un particolare campo
        nella risposta del modem
*/
bool searchField(unsigned char *s,const char *field,char *tmp)
{
    int i=0;

    tmp[0]=0;
```

```
    int l=strlen(field);

    while(s[i]) {
        if (!strncasecmp((char *) s+i,field,l)) {
            strcpy((char*) tmp,(char *) s+i+l);
            i=0;
            while(tmp[i]) {
                if (tmp[i] == 0x0D || tmp[i] == 0x0A) break;
                i++;
            }
            tmp[i]=0;
            trim(tmp);
            return true;
        }
        i++;
    }
    return false;
}

/*
        Acquisisce informazioni sul segnale GSM
*/
void modemInfo(int fd)
{
        char cmd[512];
        unsigned char buffer[1024];
        char tmp[64];

        memset(buffer,0,sizeof(buffer));
        query(fd,(unsigned char *) "at+cops?",buffer);
        writeLog((char*) "+COPS: %s",buffer);
        if (searchField(buffer,"+cops:",tmp)) {
                sprintf(cmd,"wget -qO- \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                bin/web.admin.fcgi?source=ctstrainmeters&what=cops&who=%s\" >
                /dev/null 2>&1",RDAT_USER,RDAT_PASSWORD,tmp);
                system(cmd);
                }

        memset(buffer,0,sizeof(buffer));
        query(fd,(unsigned char *) "at+creg?",buffer);
        writeLog((char*) "+CREG: %s",buffer);
        if (searchField(buffer,"+creg:",tmp)) {
                 sprintf(cmd,"wget -qO-
                 \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                 bin/web.admin.fcgi?source=ctstrainmeters&what=creg&who=%s\" >
                 /dev/null 2>&1",RDAT_USER,RDAT_PASSWORD,tmp);
                sprintf(cmd);
        }

        memset(buffer,0,sizeof(buffer));
        query(fd,(unsigned char *) "at+csq",buffer);
        writeLog((char*) "+CSQ: %s",buffer);
        if (searchField(buffer,"+csq:",tmp)) {
                sprintf(cmd,"wget -qO- \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                bin/web.admin.fcgi?source=ctstrainmeters&what=csq&who=%s\" >
                /dev/null 2>&1",RDAT_USER,RDAT_PASSWORD,tmp);
                system(cmd);
        }
}
#endif

/*
```

```
        Apre la porta seriale
*/
int open_port(void)
{
        int fd;

        fd = open(MODEM_PORT, O_RDWR | O_NOCTTY | O_NDELAY);

        if(fd == -1) fprintf(HLOG,"open_port: Unable to open %s.
      \n",MODEM_PORT);
                else
                fprintf(HLOG,"port is open.\n");
        return fd;
}

/*
        Confronta la porta seriale
*/
int configure_port(int fd)
{
        struct termios port_settings;

        cfsetispeed(&port_settings, B9600);
        cfsetospeed(&port_settings, B9600);

        port_settings.c_cflag &= ~PARENB;
        port_settings.c_cflag &= ~CSTOPB;
        port_settings.c_cflag &= ~CSIZE;
        port_settings.c_cflag |= CS8;
        port_settings.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
        port_settings.c_iflag &= ~(IXON | IXOFF | IXANY);
        port_settings.c_cc[VTIME] = 5;
        port_settings.c_cc[VMIN]  = 1;

        tcsetattr(fd, TCSANOW, &port_settings);

        return fd;
}

/*
        Resetta il modem fisicamente attraverso un'interruttore elettronico
        collegato alla porta parallela
*/
void resetModem(int &fd)
{
    static int hour=-1;

    time_t t = time(NULL);
    struct tm tm = *localtime(&t);

    if (hour < 0 || tm.tm_hour == hour) {
        if (hour < 0) {
                        writeDate();
                        printf("Close serial port\n");
                        hour=tm.tm_hour;
        }
        if (fd > 0) close(fd);
        hour += RESET_HOURS_DELAY;
        if (hour > 23) hour=0;
        writeDate(); printf("Modem OFF\n");
        outb(1,SWITCH_PORT);  // turn off modem
        sleep(3);
```

```
        outb(0, SWITCH_PORT);  // turn off modem
        writeDate(); printf("Modem ON\n");
        sleep(20);
        writeDate(); printf("Modem Ready\n");
        writeDate(); printf("Open serial port\n");
        fd = open_port();
        if (fd < 0) {
                writeDate();
                printf("Couldn't open serial port, %s\n",strerror(errno));
                return;
        }
        writeDate(); printf("Configure serial port\n");
        configure_port(fd);
        writeDate(); printf("Reset Modem next hour:%d\n",hour);
    }
}

////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                 Funzioni per il trattamento degli SMS              //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

/*
        Verifica se il carattere fa parte dell'insieme corretto
*/
bool isvalid(unsigned char c)
{
    if (c >= '0' && c <= '9') return true;
    if (c >= 'A' && c <= 'Z') return true;
    if (c >= 'a' && c <= 'z') return true;
    if (c == 0x0A || c == 0x0D ||
        c == ','  || c == '_'  || c == '+' ||
        c == '='  || c == ':'  || c == 0x20) return true;
    return false;
}

/*
        Elimina dalla stringa i caratteri non voluti
*/
void clear(unsigned char *s)
{
    int k=0;
    int i=0;

    while(s[i]) {
            if (!isvalid(s[i])) { i++; continue; }
            s[k]=s[i];
            k++;
            i++;
    }

    s[k]=0;
}

/*
        Richiede informazioni al modem attraverso un comando AT Hayes
*/
int query(int fd,unsigned char *s,unsigned char *tmp)
{
```

```
    fprintf(HLOG,"[%s]",s);
    int r;
    while(true) {
        r=read(fd,tmp,1);
        if (r < 0) {
                        if (errno == EAGAIN) break;
                        printf("[Serial Error %d,%s]",errno,strerror(errno));
                        return 0;
                    }
        if (!r) break;
    }
    int l=sprintf((char *) tmp,"%s\r\n",s);
        printf("[Send \"%s\"]",tmp);
        r=write(fd,tmp,l);
        if (r < l) {
                        if (r >= 0) printf("[Data writed %d/%d]",r,l);
                        printf("[Serial Error %d,%s]",errno,strerror(errno));
                        return 0;
        }
        int k=0;
        tmp[k]=0;
        printf("[QUERY READ]");
        while(true) {
             r=read(fd,tmp+k,1);
            if (r < 0) {
                         if (errno == EAGAIN) continue;
                         printf("[Serial Error %d,%s]",errno,strerror(errno));
                         return 0;
                       }
            if (!r)  continue;
            if (tmp[k])
            {                                                fprintf(HLOG,"|%d[%d][%s][%
            s][%d]",k,sizeof(CMS_ERROR_MESSAGE)-1,(char *) tmp+k-
            sizeof(CMS_ERROR_MESSAGE),CMS_ERROR_MESSAGE,
                              !strncasecmp((char *) tmp+k-
            sizeof(CMS_ERROR_MESSAGE),CMS_ERROR_MESSAGE,sizeof(CMS_ERROR_MESSAGE
            )-1));
                     fflush(stdout);
                     if (!strncasecmp((char *) tmp+k-
            sizeof(CMS_ERROR_MESSAGE)+2,CMS_ERROR_MESSAGE,sizeof(CMS_ERROR_MESSA
            GE)-1)) {
                                 writeLog((char*) "ERROR: %s",tmp);
                        printf("[QUERY OUT,ERROR]");
                                 return 0;
                     }
                     if (!strncasecmp((char *) tmp+k-
            sizeof(OK_MESSAGE)+2,OK_MESSAGE,sizeof(OK_MESSAGE)-1)) {
                                 writeLog((char*) "OK: %s",tmp);
                                 k++;
                                 tmp[k]=0;
                        printf("[QUERY OUT,OK]");
                                 return k;
                     }
                     k++;
            }
        }

        printf("[QUERY OUT]");
        return 0;
}

/*
```

```
        Ricerca la qualità del segnale nel SMS
*/
bool searchSignal(unsigned char *s,unsigned char *tmp)
{
    int i=0;

    tmp[0]=0;
    int l=strlen(SIGNAL_MESSAGE);
    while(s[i]) {
        if (!strncasecmp((char *) s+i,SIGNAL_MESSAGE,l)) {
            strcpy((char*) tmp,(char *) s+i+l+2);
            i=0;
            while(tmp[i]) {
                        if (tmp[i] == 0x20) tmp[i]='_';
                        i++;
            }
            return true;
        }
        i++;
    }
    return false;
}

/*
        Ricerca l'IP nel SMS
*/
bool searchIP(unsigned char *s,unsigned char *tmp)
{
    int i=0;

    tmp[0]=0;
    int l=strlen(ADDRESS_MESSAGE);
    while(s[i]) {
        if (!strncasecmp((char *) s+i,ADDRESS_MESSAGE,l)) {
            strcpy((char*) tmp,(char *) s+i+l+2);
            return true;
        }
        i++;
    }
    return false;
}

/*
        Converte una cifra esadecimale in decimale
*/
int ihex(unsigned char c)
{
    if (c >= '0' && c <= '9') return c-'0';
    if (c >= 'a' && c <= 'f') return c-'a'+10;
    return c-'A'+10;
}

/*
        Converte un numero decimale di 8 bit nella sua
        rappresentazione esadecimale
*/
int hex(unsigned char *s,int n,unsigned char *tmp)
{
    int i;
    for (i=0;i<n/2;i++) tmp[i] =  ihex(s[i*2])*16+ihex(s[i*2+1]);
    return i;
}
```

```
/*
        Estrae l'orario e la data in cui il messaggio è stato inviato
*/
void decodeTimestamp(unsigned char *s,int n,DT &dt)
{
    int i;
    char tmp[32];

    for (i=0;i<n/2;i++) { tmp[i*2]=s[i*2+1]; tmp[i*2+1]=s[i*2]; }
    tmp[12]=0;
    fprintf(HLOG,"[TIMESTAMP %s]",tmp);
    dt.year=(tmp[0]-'0')*10+tmp[1]-'0';
    dt.year += 2000;
    dt.month=(tmp[2]-'0')*10+tmp[3]-'0';
    dt.day=(tmp[4]-'0')*10+tmp[5]-'0';
    dt.hour=(tmp[6]-'0')*10+tmp[7]-'0';
    dt.min=(tmp[8]-'0')*10+tmp[9]-'0';
    dt.sec=(tmp[10]-'0')*10+tmp[11]-'0';
}

/*
        Estrae il numero di telefono del mittente
*/
void decodeNumber(unsigned char *s,int n,unsigned char *tmp)
{
    int i;
    tmp[0]='+';
    for (i=0;i<n/2;i++) { tmp[1+i*2]=s[i*2+1]; tmp[1+i*2+1]=s[i*2]; }
    tmp[n+1]=0;
}

/*
        Associa il nome della stazione al numero del mittente
*/
int decodeName(unsigned char *number,unsigned char *name)
{
    int i;

    for (i=0;i<NSTATIONS;i++)
        if (!strcasecmp((char *) number,stations[i].number)) {
                strcpy((char *) name,stations[i].name);
                return i;
        }
    name[0]=0;
    return i;
}

/*
        Converte gli octets in bytes
*/
int decodePDU(unsigned char *buffer,int n,unsigned char *tmp)
{
    int i;
    unsigned char c=0;
    int bits=OCTET_SIZE;
    int k=0;

    unsigned char mask=0x7f;

    for (i=0;i<n;i++) {
        c |= ((buffer[i] & inf) << (OCTET_SIZE-bits));
```

```
            tmp[k]=c;
            k++;
            c=buffer[i] & ~mask;
            c >>= bits;
            mask >>= 1;
            bits--;
            if (!bits) {
                tmp[k]=c;
                k++;
                bits= OCTET_SIZE;
                c=0;
                mask=0x7f;
            }
        }

        for (i=0;i<k;i++) tmp[i]=alphabet[tmp[i]];

        tmp[k]=0;

        return k;
}

/*
        Estrae dalla risposta del modem l'SMS in formato PDU
*/
int parserOctets(unsigned char *s,unsigned char *tmp)
{
        int i=0;

        tmp[0]=0;
        while(s[i]) {
           if (!strncasecmp((char *) s+i,CMGR_MESSAGE,sizeof(CMGR_MESSAGE)-1)) {
                  while(s[i] != 0x0D && s[i] != 0x0A) i++;
                  while(s[i] == 0x0D || s[i] == 0x0A) i++;
                  int k=0;
                  while(s[i] != 0x0D && s[i] != 0x0A) { tmp[k]=s[i]; k++; i++; }
                  tmp[k]=0;
                  trim((char*) tmp);
                  return k;
                }
                i++;
        }

        return 0;
}

/*
        Cancella un SMS dalla memoria del modem o della SIM
*/
void deleteSMS(int fd,int index)
{
    unsigned char buffer[1024];

    memset(buffer,0,sizeof(buffer));
    #ifndef _DISABLE_DELETE
    char s[64];

    writeLog((char*) "******************SEND DELETE
     SMS******************\n%s",buffer);
    sprintf(s,"at+cmgd=%d",index);
    query(fd,(unsigned char *) s,buffer);
    writeLog((char*) "******************DELETE
```

```
     SMS******************\n%s",buffer);
    #endif
    fflush(stdout);
}

/*
        Richiede un SMS al modem e lo processa per estrarre le informazioni
*/
bool processSMS(int fd,int index)
{
    unsigned char buffer[1024];
    unsigned char tmp[1024];
    int n;
    int i;
    unsigned char number[32];
    unsigned char ip[32];
    unsigned char signal[32];
    unsigned char name[32];
    unsigned char timestamp[32];
    char cmd[512];
    DT dt;
    int r[5];
    char s[128];

    sprintf(s,"at+cmgr=%d",index);

    memset(buffer,0,sizeof(buffer));
    if (!query(fd,(unsigned char *) s,buffer)) return false;
        writeLog((char*) "QUERY SMS: %s",buffer);
        clear(buffer);
    n=parserOctets(buffer,tmp);

    decodeTimestamp(tmp+PDU_TIMESTAMP_ADDRESS,PDU_TIMESTAMP_SIZE,dt);
    sprintf((char*)
timestamp,"%2.2d/%2.2d/%4.4d %2.2d:%2.2d:%2.2d",dt.day,dt.month,dt.year,dt.hour,
dt.min,dt.sec);
    fprintf(HLOG,"Timestamp: [%s]",timestamp);
    writeLog((char*) "TIMESTAMP: %s",timestamp);

decodeNumber(tmp+PDU_NUMBER_ADDRESS,ihex(tmp[PDU_NUMBER_ADDRESS_SIZE])*16+ihex(t
mp[PDU_NUMBER_ADDRESS_SIZE+1]),number);
    fprintf(HLOG,"Number: [%s]",number);
    writeLog((char*) "NUMBER: %s",number);
    int id=decodeName(number,name);
    if (id == NSTATIONS) return true;
    fprintf(HLOG,"Name: [%s %d]",name,id);
    sprintf(s,"NAME: %s %d",name,id);
    writeLog((char*) "%s",(unsigned char*) s);
    fprintf(HLOG,"n:%d[%s]\n",n,tmp+PDU_DATA_ADDRESS);
    n -= PDU_DATA_ADDRESS;
    int
length=ihex(tmp[PDU_DATA_ADDRESS_SIZE])*16+ihex(tmp[PDU_DATA_ADDRESS_SIZE+1]);
    n=hex(tmp+PDU_DATA_ADDRESS,n,buffer);
    fprintf(HLOG,"[");
    for(i=0;i<n;i++) fprintf(HLOG,"%2.2hX",buffer[i]);
    fprintf(HLOG,"]\n");
    fprintf(HLOG,"_____________\n");

    n=decodePDU(buffer,n,tmp);
    tmp[length]=0;
    for(i=0;i<n;i++) fprintf(HLOG,"%d ",tmp[i]);
    fprintf(HLOG,"_____________\n");
```

```
tmp[n]=0;
fprintf(HLOG,"\nmessage: %s\n",tmp);
printf("[SEARCH IP");
if (searchSignal(tmp,signal)) {
    if (stations[id].dt <= dt) {
                    stations[id].dt=dt;
                    strcpy(stations[id].signal,(char *)signal);
                    #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
                    if (strcmp((char *) name,CNE_NAME)) {
                            sprintf(cmd,"wget -qO-
                    \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                    bin/web.admin.fcgi?source=ctstrainmeters&what=signal&who=
                    %s:%s:%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d\" > /dev/null 2>&1",

                    RDAT_USER,RDAT_PASSWORD,name,signal,dt.year,dt.month,dt.d
                    ay,dt.hour,dt.min,dt.sec);
                            system(cmd);
                            }
                    #endif
    }
    else
    {
    #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
    if (strcmp((char *) name,CNE_NAME)) {
                            sprintf(cmd,"wget -qO-
             \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
             bin/web.admin.fcgi?source=ctstrainmeters&what=old_signal&who=%s
             :%s:%4.4d%2.2d%2.2d%2.2d%2.2d%2.2d\" > /dev/null 2>&1",

                    RDAT_USER,RDAT_PASSWORD,name,signal,dt.year,dt.month,dt.d
             ay,dt.hour,dt.min,dt.sec);
            system(cmd);
            }
    #endif
    }

    deleteSMS(fd,index);
    return true;
} else
if (searchIP(tmp,ip)) {
     printf("[IP FOUND]");
    if (stations[id].dt <= dt) {
                        stations[id].dt=dt;
                    strcpy(stations[id].ip,(char *)ip);
    }
    else
    {
            #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
            if (strcmp((char *) name,CNE_NAME)) {
            sprintf(cmd,"wget -qO-
            \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
            bin/web.admin.fcgi?source=ctstrainmeters&what=old_ip&who=%s:%s:%
            4.4d%2.2d%2.2d%2.2d%2.2d%2.2d\" > /dev/null 2>&1",

                    RDAT_USER,RDAT_PASSWORD,name,ip,dt.year,dt.month,dt.day,d
            t.hour,dt.min,dt.sec);
                    system(cmd);
            }
            #endif
            deleteSMS(fd,index);
            return true;
    }
```

```
        #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
        if (strcmp((char *) name,CNE_NAME)) {
                printf("[RDAT IP]");
                sprintf(cmd,"wget -qO- \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
bin/web.admin.fcgi?source=ctstrainmeters&what=ip&who=%s:%s:%4.4d%2.2d%2.2d%2.2d%
2.2d%2.2d\" > /dev/null 2>&1",

        RDAT_USER,RDAT_PASSWORD,name,ip,dt.year,dt.month,dt.day,dt.hour,dt.min,d
t.sec);
                system(cmd);
                // Test router www
                printf("[Test Router]");
                sprintf(cmd,"nmap -p 25328 %s | egrep 'open'",ip);
                r[0]=system(cmd);
                // Test Shoebox www
                sprintf(cmd,"nmap -p 32454 %s | egrep 'open'",ip);
                r[1]=system(cmd);
                // Test iboot www
                sprintf(cmd,"nmap -p 54235 %s | egrep 'open'",ip);
                r[2]=system(cmd);
                printf("[Test Rabbit]");
                sprintf(cmd,"nmap -p 25000 %s | egrep 'open'",ip);
                printf("[%s]",cmd);
                r[3]=system(cmd);
                printf("[Test SSH]");
                sprintf(cmd,"nmap -p 6583 %s | egrep 'open'",ip);
                printf("[%s]",cmd);
                r[4]=system(cmd);
                printf("[RDAT Link]");
                sprintf(cmd,"wget -qO- \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                bin/web.admin.fcgi?source=ctstrainmeters&what=link&who=%s:%d,%d,
                %d,%d,%d\" > /dev/null 2>&1",
                        RDAT_USER,RDAT_PASSWORD,name,r[0],r[1],r[2],r[3],r[4]);
                system(cmd);
        }
        #endif

        fprintf(HLOG,"IP: [%s]",ip);
        writeLog((char*) "IP: %s",ip);
        char filename[128];
        sprintf(filename,"%s/%s/ip",HOME_DIRECTORY,name);
        FILE *handle=fopen(filename,"wt+");
        fprintf(handle,"%s\n",ip);
        fclose(handle);
        #ifdef ENABLE_LOCAL_DNS
        printf("[hosts]");
        system("/var/ddns/hosts.sh");
        #endif
        #ifdef ENABLE_NAT
        printf("[iptables]");
        system("/var/ddns/iptables.sh");
        #endif
        #ifdef ENABLE_WEB_PAGES
        writeRedirect(name,ip);
        printf("[rsync]");
                sprintf(cmd,"rsync -avz -e ssh
         /var/ddns %s@magftp.ct.ingv.it:. >>%s/ddns.ssh.log
         2>&1",STRAIN_USER,HOME_DIRECTORY);
        system(cmd);
        printf("[ps]");
          sprintf(cmd,"ps -ef | grep %s | grep bash | awk '{ print $2 }' | xargs
         kill",name);
```

```
        system(cmd);
        #endif
        #ifdef ENABLE_DOWNLOAD
        printf("[syncStrain]");
                sprintf(cmd,"su strain -c \"nohup /home/strain/Shoebox/syncStrain
         ddns %s >/dev/null 2>&1 &disown\"&",name);
                system(cmd);
        printf("[synStrain ends]");
        #endif

        #ifdef ENABLE_NSUPDATE
                nsupdate(name,ip);
        #endif
    }
    deleteSMS(fd,index);
    return true;
}

/*
        Legge tutti gli SMS presenti nella memoria del modem o della SIM
*/
void readAllMessages(int fd)
{
    unsigned char buffer[1024];
    unsigned char tmp[1024];
    char s[512];
    int storage,fill,all;
    int last;
    int j=0;
    bool found;

    int i;
    int k;

    writeLog((char*) "\n\n******************Read All Messages
     ******************\n",(unsigned char *)"");

    memset(buffer,0,sizeof(buffer));
    query(fd,(unsigned char *) "at+cpms?",buffer);
    writeLog((char*) "QUERY STORAGES: %s",buffer);

    do {
    found=false;
    i=0;
    bool end=false;
    while(!end) {
        while(buffer[i] && buffer[i] != '"')  i++;
        if (!buffer[i]) break;
        storage=i;
        i++;
        while(buffer[i] && buffer[i] != '"')  i++;
        if (!buffer[i]) break;
        i++;
        buffer[i]=0;
        i++;
        last=i;
        while(buffer[i] && buffer[i] != ',')  i++;
        if (!buffer[i]) break;
        buffer[i]=0;
        if (sscanf((char*) buffer+last,"%d",&fill) != 1) return;
        i++;
        last=i;
```

```
        while(buffer[i] && buffer[i] != ',')  i++;
        if (!buffer[i]) end=true;
        buffer[i]=0;
        if (sscanf((char*) buffer+last,"%d",&all) != 1) return;
        i++;
        fprintf(HLOG,"[CPMS %s %d %d]",buffer+storage,fill,all);
        if (j > 0 && fill) {
                found=true;
                sprintf(s,"at+cpms=%s",buffer+storage);
                query(fd,(unsigned char *) s,tmp);
                for (k=0;k<all;k++) processSMS(fd,k+1);
        }
        j++;
    }
    } while(found);
}


////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//           Funzioni di supporto all'esecuzione come daemon          //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

#ifdef _ENABLE_DAEMON
/*
        Trasforma il processo in un daemon per l'esecuzione in background
        delle operazioni e l'avvio durante il boot del sistema
*/
void daemon(void)
{
    pid_t pid;

    pid = fork ();
    if (pid == -1) exit(EXIT_FAILURE);
        else if (pid != 0)
            exit (EXIT_SUCCESS);

    if (setsid ( ) == -1) exit(EXIT_FAILURE);

    if (chdir (HOME_DIRECTORY) == -1) return exit(EXIT_FAILURE);

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    char filename[256];
    sprintf(filename,"%s/%s ",HOME_DIRECTORY,LOG_FILE);
    open("/dev/null",O_RDWR);
    open(filename,O_RDWR);
    open(filename,O_RDWR);
    HLOG=fopen(filename,"at+");

    FILE *handle=fopen(PID_FILE,"wt+");
    fprintf(handle,"%d\n",getpid());
    fclose(handle);

    setvbuf(stdout, NULL, _IONBF, 0);
}
#endif
```

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                                                                    //
//                  Punto d'ingresso del programma (Main)             //
//                                                                    //
//                                                                    //
////////////////////////////////////////////////////////////////////////

int main(int argc,char *argv[])
{
    int fd=-1;
    char cmd[512];
    #ifdef _ENABLE_DAEMON
        daemon();
    #else
        setvbuf(stdout, NULL, _IONBF, 0);
    #endif

    #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
    int delay=0;
    #endif

            if(ioperm(SWITCH_PORT,1,1)) { printf("Couldn't open parallel port");
        return -1; }

        while(true) {
            resetModem(fd);
            if (fd < 0) return -1;
            writeDate();
            printf("Read all SMS\n");
            readAllMessages(fd);
            writeDate();
            printf("sleep\n");
            sleep(SMS_DELAY_TIME);
            #ifdef ENABLE_REMOTE_DIAGNOSTIC_AND_ADMINISTRATION_TOOLS
            if (!delay) {
                        delay=0;
                         sprintf(cmd,"wget -qO-
                         \"http://%s:%s@magftp.ct.ingv.it:2016/cgi-
                         bin/web.admin.fcgi?source=ctstrainmeters&what=ddns&who=i
                         dle\" > /dev/null 2>&1",RDAT_USER,RDAT_PASSWORD);
                        system(cmd);
                        modemInfo(fd);
            }
            delay++;
            if (delay == IDLE_DELAY_TIME/SMS_DELAY_TIME) delay=0;
            #endif
        }
    #ifdef _ENABLE_DAEMON
    unlink(PID_FILE);
    fclose(HLOG);
    #endif
        return 0;
}
```

Appendice A3. Codice sorgente: ddns.sh

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          ddns
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Example initscript
# Description:       This file should be used to construct scripts to be
#                    placed in /etc/init.d.
### END INIT INFO

# Do NOT "set -e"

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC="Dynamic DNS through GSM Short Message"
NAME=ddns
DAEMON=/usr/sbin/$NAME
DAEMON_ARGS="--options args"
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.2-14) to ensure that this file is present
# and status_of_proc is working.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
{
        # Return
        #   0 if daemon has been started
        #   1 if daemon was already running
        #   2 if daemon could not be started
        start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON --
test > /dev/null \
                || return 1
        start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON -- \
                $DAEMON_ARGS \
                || return 2
        # Add code here, if necessary, that waits for the process to be ready
        # to handle requests from services started subsequently which depend
        # on this one.  As a last resort, sleep for some time.
}

#
# Function that stops the daemon/service
#
```

```
do_stop()
{
        # Return
        #   0 if daemon has been stopped
        #   1 if daemon was already stopped
        #   2 if daemon could not be stopped
        #   other if a failure occurred
        start-stop-daemon --stop --quiet --retry=TERM/30/KILL/5 --pidfile
         $PIDFILE --name $NAME
        RETVAL="$?"
        [ "$RETVAL" = 2 ] && return 2
        # Wait for children to finish too if this is a daemon that forks
        # and if the daemon is only ever run from this initscript.
        # If the above conditions are not satisfied then add some other code
        # that waits for the process to drop all resources that could be
        # needed by services started subsequently.  A last resort is to
        # sleep for some time.
        start-stop-daemon --stop --quiet --oknodo --retry=0/30/KILL/5 --exec
         $DAEMON
        [ "$?" = 2 ] && return 2
        # Many daemons don't delete their pidfiles when they exit.
        rm -f $PIDFILE
        return "$RETVAL"
}

#
# Function that sends a SIGHUP to the daemon/service
#
do_reload() {
        #
        # If the daemon can reload its configuration without
        # restarting (for example, when it is sent a SIGHUP),
        # then implement that here.
        #
        start-stop-daemon --stop --signal 1 --quiet --pidfile $PIDFILE --name
$NAME
        return 0
}

case "$1" in
  start)
        [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
        do_start
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  stop)
        [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
        do_stop
        case "$?" in
                0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
                2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
        esac
        ;;
  status)
        status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
        ;;
  #reload|force-reload)
        #
        # If do_reload() is not implemented then leave this commented out
```

```
        # and leave 'force-reload' as an alias for 'restart'.
        #
        #log_daemon_msg "Reloading $DESC" "$NAME"
        #do_reload
        #log_end_msg $?
        #;;
        restart|force-reload)
        #
        # If the "reload" option is implemented then remove the
        # 'force-reload' alias
        #
        log_daemon_msg "Restarting $DESC" "$NAME"
        do_stop
        case "$?" in
          0|1)
                do_start
                case "$?" in
                        0) log_end_msg 0 ;;
                        1) log_end_msg 1 ;; # Old process is still running
                        *) log_end_msg 1 ;; # Failed to start
                esac
                ;;
          *)
                # Failed to stop
                log_end_msg 1
                ;;
        esac
        ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-reload}" >&2
        exit 3
        ;;
esac

:
```

Appendice A4. Codice sorgente: hosts.sh

```
#!/bin/sh
cp -f /etc/hosts_default /etc/hosts
((cat /var/ddns/druv/ip | tr -d '\n'); echo " druv") >> /etc/hosts
((cat /var/ddns/degi/ip | tr -d '\n'); echo " degi") >> /etc/hosts
((cat /var/ddns/dpdn/ip | tr -d '\n'); echo " dpdn") >> /etc/hosts
((cat /var/ddns/dmsc/ip | tr -d '\n'); echo " dmsc") >> /etc/hosts
((cat /var/ddns/cne/ip | tr -d '\n'); echo " cne") >> /etc/hosts
```

Appendice A5. Codice sorgente: Makefile

```
all:    ddns

ddns: ddns.cpp ddns.sh
        gcc -D_ENABLE_DAEMON -o ddns ddns.cpp
        -killall ddns
        -/etc/init.d/ddns stop
        -mkdir /var/ddns
        -mkdir /var/ddns/degi
        -mkdir /var/ddns/druv
        -mkdir /var/ddns/dpdn
        -mkdir /var/ddns/dmsc
        -mkdir /var/ddns/cne
        chmod 777 /var/ddns
        chmod 777 /var/ddns/degi
        chmod 777 /var/ddns/druv
        chmod 777 /var/ddns/dpdn
        chmod 777 /var/ddns/dmsc
        chmod 777 /var/ddns/cne
        cp -f ddns.sh /etc/init.d/ddns
        cp -f ddns /usr/sbin/ddns
        cp -f hosts.sh /var/ddns/.
        cp -f iptables.sh /var/ddns/.
        -ln -s ../init.d/ddns /etc/rc2.d/S99ddns
        /etc/init.d/ddns start
        ps -aux | grep ddns

clean:
        rm -f ddns
        rm -f ddns.o
```

Appendice A6. Codice sorgente: iptables.sh

```
#!/bin/sh
echo "Stopping firewall and allowing everyone..."
IP_CNE="$(cat /var/ddns/cne/ip 2> /dev/null | tr -d '\n')"
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -I INPUT -p tcp --dport 1723 -m state --state NEW -j ACCEPT
iptables -I INPUT -p gre -j ACCEPT
iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE
iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -s 192.168.0.0/24 -j TCPMSS
--clamp-mss-to-pmtu
if [ ! -z "$IP_CNE" ]; then
        echo "Set CNE IP"
        iptables -i eth0 -v -t nat -A PREROUTING -d $IP_CNE -j NETMAP --to
192.168.0.238
        iptables -i ppp0 -v -t nat -A PREROUTING -d 192.168.0.238 -j NETMAP --to
$IP_CNE
fi
```

ISSN 1590-2595

Quaderni di Geofisica

http://istituto.ingv.it/l-ingv/produzione-scientifica/quaderni-di-geofisica/

I Quaderni di Geofisica coprono tutti i campi disciplinari sviluppati all'interno dell'INGV, dando particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari, che per tipologia e dettaglio necessitano di una rapida diffusione nella comunità scientifica nazionale ed internazionale. La pubblicazione on-line fornisce accesso immediato a tutti i possibili utenti. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-7941

rapporti tecnici INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/rapporti-tecnici-ingv/

I Rapporti Tecnici INGV pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico e di rilevante interesse tecnico-scientifico per gli ambiti disciplinari propri dell'INGV. La collana Rapporti Tecnici INGV pubblica esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. L'Editorial Board multidisciplinare garantisce i requisiti di qualità per la pubblicazione dei contributi.

ISSN 2039-6651

Miscellanea INGV

http://istituto.ingv.it/l-ingv/produzione-scientifica/miscellanea-ingv/

La collana Miscellanea INGV nasce con l'intento di favorire la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV (sismologia, vulcanologia, geologia, geomagnetismo, geochimica, aeronomia e innovazione tecnologica). In particolare, la collana Miscellanea INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli ecc..

Coordinamento editoriale e impaginazione
Centro Editoriale Nazionale | INGV

Progetto grafico e redazionale
Daniela Riposati | Laboratorio Grafica e Immagini | INGV

Via di Vigna Murata, 605
00143 Roma
Tel. +39 06518601 Fax +39 065041181

http://www.ingv.it

Istituto Nazionale di Geofisica e Vulcanologia