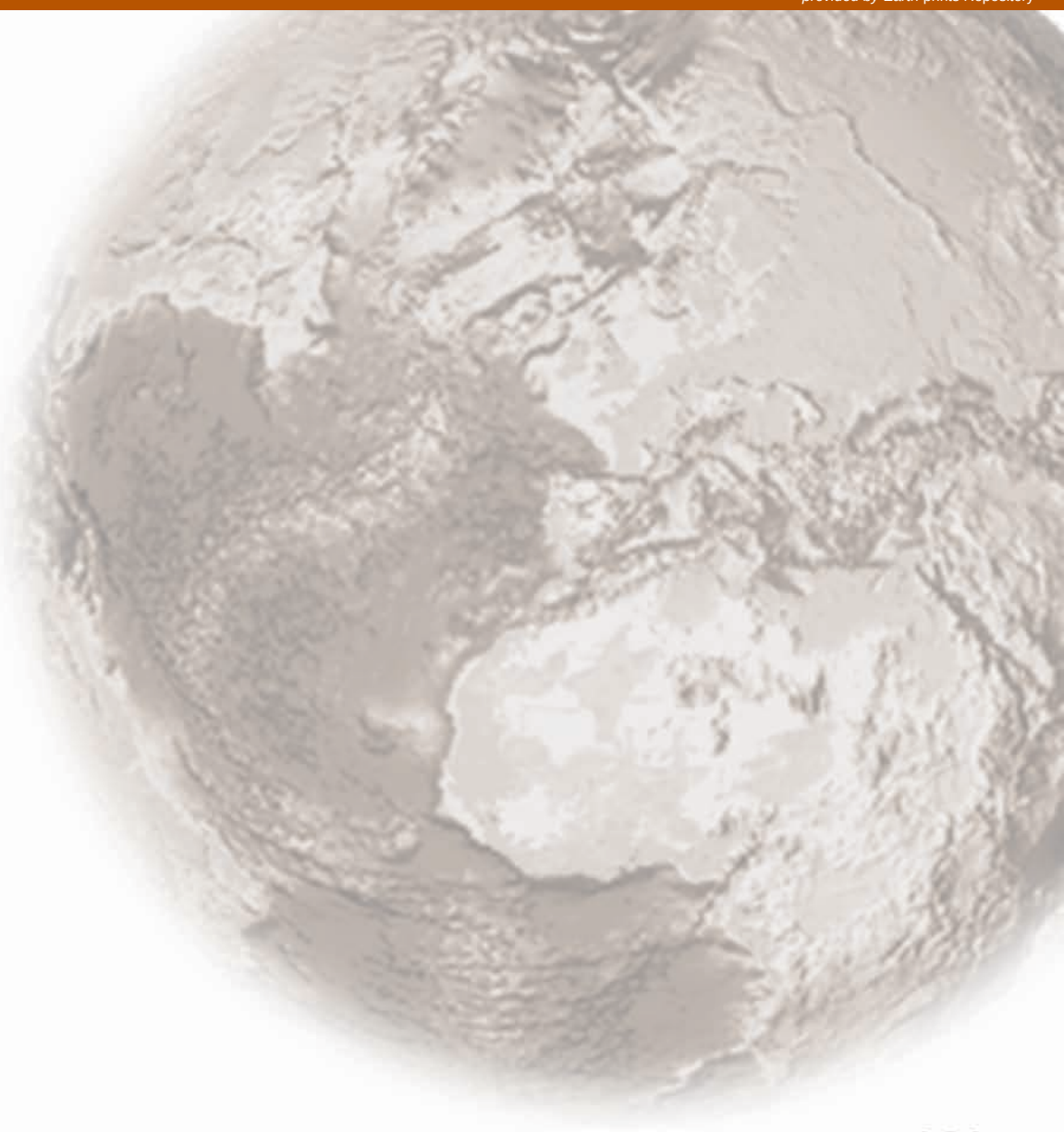
Rapporti Tecnici INGV



2007

libnmxp e nmxptool: software Open-Source per trasmissioni dati sismici Nanometrics

Matteo Quintiliani

n.51

Istituto Nazionale di Geofisica e Vulcanologia
Via di Vigna Murata 605 - 00143 Roma
tel 06518601 • fax 065041181
www.ingv.it

LIBNMXP E NMXPTOOL: SOFTWARE OPEN-SOURCE PER TRASMISSIONI DATI SISMICI NANOMETRICS

Matteo Quintiliani

Istituto Nazionale di Geofisica e Vulcanologia
Centro Nazionale Terremoti
quintiliani@ingv.it

Introduzione

Il presente documento descrive le modalità di impiego della libreria software progettata dall'autore al fine di implementare i protocolli di trasmissione Nanometrics. Lo sviluppo di tale libreria nasce principalmente dall'esigenza all'interno dell'INGV di gestire un numero sempre più crescente di canali sismici acquisiti tramite sistema Nanometrics. La libreria denominata ***libnmxp*** offre un insieme di API (Application Program Interface) ben documentate che permettono di sviluppare software capace di interagire con i due tipi di server Nanometrics:

- *NaqsServer* il quale implementa il protocollo per trasmissioni di dati in tempo reale;
- *DataServer* il quale implementa il protocollo per il recupero di dati archiviati.

Insieme alla libreria viene inoltre distribuito un programma chiamato ***nmxptool*** che basandosi su di essa, permette di eseguire interrogazioni, ricevere dati in tempo reale e/o off-line, ed inoltre permette di salvare questi ultimi in diversi formati, quali NMX e mini-SEED. Tale programma può inoltre essere utilizzato come modulo per il sistema ***Earthworm*** o come plug-in per *server* ***SeedLink***.

Uno dei principali contributi offerti da questo sviluppo consiste nella possibilità di gestire connessioni di tipo *Raw Stream* con riordinamento dei pacchetti ritrasmessi: ciò permette di garantire un buon compromesso fra la continuità del dato e una bassa latenza.

L'intero sviluppo si è basato sul manuale del corso Nanometrics [Nanometrics, Inc., 1989-2002], in particolare su *Nanometrics Data Formats, Reference Guide* inclusa nella sezione *Software Reference Manuals*.

La libreria ***libnmxp*** e il programma ***nmxptool*** sono scritti in linguaggio C e sviluppati usando i *GNU Build Tools* (*automake*, *autoconf*, e script *configure*) tenendo in considerazione gli aspetti di compilazione trasversale (*cross-compilation*) su tutte le piattaforme di tipo POSIX/UNIX. I sorgenti sono gratuiti e possono essere modificati e ridistribuiti sotto i termini *GNU Library General Public License,* ulteriori informazioni possono essere trovate su http://www.gnu.org/.

1. Protocolli Nanometrics

Prendendo ad esempio una configurazione tipica del flusso trasmissivo dei dati da una stazione sismica, acquisita attraverso i *server* Nanometrics, fino a raggiungere i processi di acquisizione e localizzazione situati nella Sala di Monitoraggio dell'INGV, come mostrato in figura 1 possiamo suddividere tale flusso in due parti:

1. ***Stazione Sismica - Nanometrics Server***: i dati ricevuti dalla porta seriale di uno strumento vengono convertiti nel formato *NMXP* e poi spediti in pacchetti *UDP* ai *server* di acquisizione.
2. ***Nanometrics Server – Client***: applicazioni software si connettono ai *server* Nanometrics per recuperare dati in tempo reale o in differita, ricevere informazioni sullo "stato di salute" (*state-of-health*) degli strumenti, *trigger* o eventi sismici.

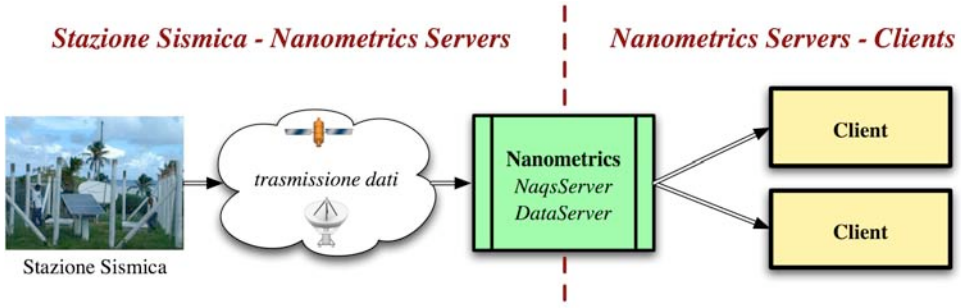


Figura 1. *Configurazione tipica di un flusso trasmissivo dei dati da una stazione sismica ai processi di acquisizione e localizzazioni situati nella Sala di Monitoraggio dell'INGV.*

Ciò di cui terremo conto in questo documento farà riferimento principalmente al lato trasmissivo *Nanometrics Server - Client* ed in particolare alle specifiche dei protocolli:

- ***Private Data Stream*** *versione 1.4*, il quale definisce il protocollo di comunicazione di un client con un *NaqsServer*.
- ***Data Access Protocol*** *versione 1.0*, il quale definisce il protocollo di comunicazione con un *DataServer*.

La differenza significativa fra i due protocolli è che ad un *NaqsServer* ci si connette per ricevere dati, informazioni sui canali, *trigger* e eventi in tempo reale (*online*) mentre ad un *DataServer* ci si connette per accedere ai dati e alle informazioni del passato (*offline*).

Di seguito, senza alcuna pretesa di completezza, vengono descritti i comportamenti generali di questi due tipi di *server* e dei rispettivi protocolli di trasmissione. Per maggiori dettagli fare riferimento al manuale del corso Nanometrics [Nanometrics, Inc., 1989-2002], in particolare *Nanometrics Data Formats, Reference Guide* inclusa nella sezione *Software Reference Manuals*.

1.1 Private Data Stream

Il NaqsServer fornisce accesso online via TCP/IP a dati di tipo *time-series*, *serial data*, *trigger*, e *state-of-health*. Il sottosistema del NaqsServer chiamato Stream Manager si comporta come un DataServer: accetta connessioni e richieste di dati da programmi client e redirige i dati richiesti ad ogni programma client in tempo quasi reale. I dati compressi possono essere richiesti in due modi:

- *Raw stream*: tutti i pacchetti (sia quelli originali quelli ritrasmessi) sono rediretti nello stesso ordine nel quale sono ricevuti. I pacchetti possono essere persi, duplicati, ritrasmessi, non ordinati, ma con minimo ritardo.
- *Buffered stream*: anche chiamato *Short-term-complete data stream*. Per ogni canale viene garantito l'ordine cronologico dei pacchetti ma possono riscontrarsi piccoli *gap* temporali ogni

qualvolta si verifichi una ritrasmissione di un pacchetto dal lato trasmissivo *Stazione Sismica – Nanometrics Server*.

Ogni programma che abbia necessità di interagire con un NaqsServer deve implementare il protocollo di comunicazione *Private Data Stream versione 1.4* così come descritto schematicamente di seguito:

1. Aprire un socket su Stream Manager. La porta di default è la 28000.
2. Inviare un messaggio di tipo *Connect* allo Stream Manager.
3. Ricevere e gestire il messaggio di tipo *ChannelList* dallo Stream Manager.
4. (opzionale) Inviare un messaggio di *RequestPending* finché la richiesta non è pronta. Il server attende al massimo 30 secondi, dopodiché chiude la connessione se non ha ricevuto né una Request Pending, né un messaggio di *AddChannels*.
5. Inviare un messaggio di tipo *AddChannels* allo Stream Manager.
6. Ricevere fino alla fine i pacchetti dallo Stream Manager ed elaborarli. Opzionalmente inviare nuovi messaggi *AddChannels* (passo 5) oppure *RemoveChannels*. Il client deve saper gestire i messaggi di tipo *Error*.
7. Inviare un messaggio di tipo *TerminateSubscription* allo Stream Manager.
8. Chiudere il socket.

1.2 Data Access Protocol

Il DataServer fornisce accesso locale e remoto via TCP/IP a dati di tipo *time-series*, *serial data*, *trigger* e *state-of-health*. Il DataServer fornisce inoltre informazioni sulla disponibilità dei dati di ogni canale per mezzo di due tipi di liste:

- *Channel List*: una lista dei canali disponibili.
- *Precis List*: una lista dei canali disponibili inclusi i tempi di inizio e di fine dei dati disponibili per ogni canale.

Ogni programma che abbia necessità di interagire con un DataServer deve implementare il protocollo di comunicazione *Data Access Protocol versione 1.0* così come descritto schematicamente di seguito:

1. Aprire un socket sul DataServer. La porta di default è la 28002.
2. Ricevere come intero di 4 byte rappresentante il tempo della connnessione sul DataServer.
3. Inviare un messaggio di tipo *ConnectRequest* includendo il tempo di connessione ricevuto precedentemente.
4. Aspettare un messaggio di tipo *Ready* dal DataServer.
5. Inviare un messaggio di tipo *Request* al DataServer.
6. Ricevere ed elaborare i dati di risposta finché non si riceve il messaggio di tipo *Ready*. Il messaggio *Ready* indica che il server è pronto per ricevere nuove richieste.
7. Ripetere i passi 5 e 6 per ogni richiesta.
8. (opzionale) Inviare un messaggio di tipo *Terminate* al DataServer.
9. Chiudere il socket.

2. La libreria libnmxp

Dopo aver descritto in generale i protocolli di comunicazione Nanometrics passiamo ora ad illustrare come la libreria è organizzata e quali sono le strutture dati e le funzioni che espone per il loro utilizzo nello sviluppo di un programma che debba interagire con un NaqsServer, un DataServer o entrambi.

La libreria è stata scritta in linguaggio C con una strutturazione a livelli dei sorgenti. In fig. 2 viene mostrata l'organizzazione logica dei sorgenti.

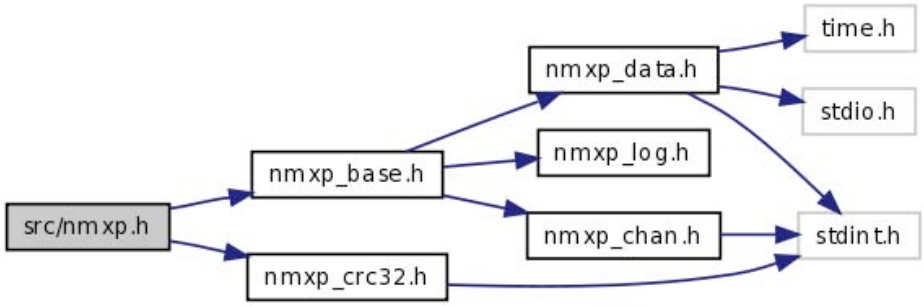


Figura 2. *Organizzazione dei sorgenti della libreria* ***libnmxp***.

Le *API* (*Application Program Interface*) che compongono la libreria offrono principalmente funzionalità a livello applicativo per lo sviluppo di software che implementi i protocolli *Private Data Stream 1.4* e *Data Access Protocol 1.0*.

Esse sono state concepite nell'ottica della realizzazione di programmi in grado di:

- manipolare i dati di tipo Nanometrics;
- richiedere, ricevere ed interpretare i dati *online* e *offline*;
- analizzare ed eseguire calcoli in tempo reale sul flusso continuo dei dati;
- recuperare e convertire *on-the-fly* i dati in diversi formati, (ad esempio mini-SEED records);
- redirezionare i dati in *server* o sistemi di altro tipo, (ad esempio *SeedLink* o *Earthworm*).

Al momento la libreria è in grado di trattare i dati di tipo *time-series* e non quelli di tipo *serial data*, *trigger* e *state-of-healt*. Per quest'ultimi si è rimandato lo sviluppo ad un futuro prossimo.

2.1 Installazione

La libreria *libnmxp* e il tool *nmxptool* sono stati sviluppati utilizzando i *GNU Build Tools* (*automake* e *autoconf*) tenendo conto degli aspetti di compilazione trasverale (*cross-compilation*) per tutte le possibili piattaforme di tipo POSIX/UNIX. Di seguito la tabella 1 mostra su quali sistemi operativi e architetture si è eseguito il test di funzionamento, la 'X' determina che il test ha avuto esito positivo.

	Little Endian		*Big Endian*	
	Intel 32-bit	**Intel 64-bit**	**SPARC 64bit**	**PowerPC**
Linux	X	X		
Solaris	X		X	
Mac OS X	X			X
FreeBSD		X		

Tabella 1. *Sistemi operativi e architetture sui quali libnmxp e nmxptool sono stati installati ed eseguiti con successo.*

I sorgenti, la documentazione e gli *scripts* di installazione della libreria e del programma vengono rilasciati in distribuzioni compresse, con nome del tipo *libnmxp-1.1.2.tar.gz*. I requisiti per l'installazione sono:

- Piattaforma POSIX
- Compilatore C GNU
- Programma make GNU

Il modo più semplice per compilare i sorgenti è:

1. `*cd*` nella directory che contiene lo script configure
2. Lanciare il comando *./configure*
3. Se *configure* termina con esito positivo allora lanciare il comando *make* per la compilazione
4. Lanciare il comando *make install* per l'installazione

Quindi, a titolo di esempio, ecco la sequenza dei comandi da eseguire in una shell per compilare *libnmxp* e *nmxptool* contenuti nella distribuzione *libnmxp-1.1.2.tar.gz*:

```
kyuzo:~ mtheo$ tar xvfz libnmxp-1.1.2.tar.gz
kyuzo:~ mtheo$ cd libnmxp-1.1.2
kyuzo:~/libnmxp-1.1.2 mtheo$ ./configure
…
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
configure:
      After running make and make install you will be able
      to compile nmpxtool into the subdirectory tools/nmxptool.
      nmxptool is a tool that implements the following protocols:
                - Nanometrics Data Access Protocol 1.0
                - Nanometrics Private Data Stream  1.4

kyuzo:~/libnmxp-1.1.2 mtheo$ make
kyuzo:~/libnmxp-1.1.2 mtheo$ su root
kyuzo:~/libnmxp-1.1.2 root# make install
kyuzo:~/libnmxp-1.1.2 root# exit
kyuzo:~/libnmxp-1.1.2 mtheo$ cd tools/nmxptool
kyuzo:~/libnmxp-1.1.2/tools/nmxptool mtheo$ ./configure
kyuzo:~/libnmxp-1.1.2/tools/nmxptool mtheo$ make
kyuzo:~/libnmxp-1.1.2/tools/nmxptool mtheo$ su root
kyuzo:~/libnmxp-1.1.2/tools/nmxptool root# make install
```

Lo script *configure* automaticamente rileva e compila se presenti: la libreria per il salvataggio dei dati in mini-SEED, i sorgenti con le funzioni base di un plug-in SeedLink e i file oggetto (i file di tipo *.o*) della libreria di Earthworm. Le compilazioni di queste tre funzionalità a supporto di *nmxptool* possono essere inibite passando rispettivamente al *configure* i seguenti tre parametri:

```
  --without-libmseed      disable support for libmseed
  --without-seedlink      disable support for seedlink
  --without-ew            disable support for earthworm
```

Per configurare *nmxptool* come modulo Earthworm bisognerà copiare i file *nmxptool.d* e *nmxptool.desc* nella directory dei parametri di Earthworm e poi modificarli secondo le proprie esigenze. La copia sarà un comando del tipo:

```
kyuzo:~/libnmxp-1.1.2/tools/nmxptool mtheo$ cp earthworm/nmxptool.* ${EW_PARAMS}
```

Per poter configurare *nmxptool* come plug-in all'interno di SeisComP sarà sufficiente copiare la directory *135_nmxptool* all'interno dei templates di SeisComP. Supponendo la *SeisComP Root* uguale a */home/sysop/seiscomp*, ecco un esempio del comando da lanciare:

```
kyuzo:~/libnmxp-1.1.2/tools/nmxptool mtheo$ cp -r \
              seiscomp_templates/135_nmxptool \
              /home/sysop/seiscomp/acquisition/templates/source/
```

Successivamente sarà possibile configurare il *plug-in* per mezzo della configurazione standard di SeisComP, ovvero lanciando il comando:

```
kyuzo:~ mtheo$ seiscomp config
```

2.2 Documentazione

La documentazione della libreria è stata redatta in lingua inglese utilizzando dei *meta-tag* all'interno del codice e per mezzo del programma ***doxygen***. Viene rilasciata all'interno del pacchetto di distribuzione nei formati *html*, *rtf* e *man*. Segue la documentazione della struttura NMXP_DATA_PROCESS e l'elenco e le relative descrizioni delle sole funzioni, quelle di più alto livello, contenute in *nmxp.h*. Per maggiori dettagli si faccia riferimento alla documentazione stessa.

Documentazione della struttura dati NMXP_DATA_PROCESS per lo sviluppo con libnmxp.

NMXP_DATA_PROCESS Struct Reference

Parameter structure for functions that process data.

#include <nmxp_data.h>

Public Attributes

1. int32_t key
   Channel Key.
2. char network [NETWORK_LENGTH]
   Network code.
3. char station [STATION_LENGTH]
   Station code.
4. char channel [CHANNEL_LENGTH]
   Channel code.
5. int32_t packet_type
   Packet type.
6. int32_t x0
   First sample. It is significant only if x0n_significant != 0.
7. int32_t xn
   Last sample. It is significant only if x0n_significant != 0.
8. int32_t x0n_significant
   Declare if xn significant.
9. int32_t oldest_seq_no
   Oldest Sequence number.
10. int32_t seq_no
    Sequence number.
11. double time
    Time first sample. Epochs.
12. void * buffer
    Nanometrics packet data.
13. int32_t length
    Packet length.
14. int * pDataPtr
    Array of samples.
15. int32_t nSamp
    Number or samples.
16. int32_t sampRate
    Sample rate.

int nmxp_sendConnect (int *isock*)

Sends the message "Connect" on a socket.

Parameters:

isock A descriptor referencing the socket.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_sendTerminateSubscription (int *isock*, NMXP_SHUTDOWN_REASON *reason*, char * *message*)

Sends the message "TerminateSubscription" on a socket.

Parameters:

isock A descriptor referencing the socket.
reason Reason for the shutdown.
message String message. It could be NULL.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_receiveChannelList (int *isock*, NMXP_CHAN_LIST ** *pchannelList*)

Receive message "NMXP_CHAN_LIST" from a socket.

Parameters:

isock A descriptor referencing the socket.
pchannelList List of channels. It will need to be freed!

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_sendAddTimeSeriesChannel (int *isock*, NMXP_CHAN_LIST * *channelList*, int32_t *shortTermCompletion*, int32_t *out_format*, NMXP_BUFFER_FLAG *buffer_flag*)

Sends the message "AddTimeSeriesChannels" on a socket.

Parameters:

isock A descriptor referencing the socket.
channelList List of channel.
shortTermCompletion Short-term-completion time = s, 1<= s <= 300 seconds.
out_format Output format. -1 Compressed packets. 0 Uncompressed packets. 0 < out_format, requested output sample rate.
buffer_flag Server will send or not buffered packets.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

NMXP_DATA_PROCESS* **nmxp_receiveData (int *isock*, NMXP_CHAN_LIST * *channelList*, const char * *network_code*)**

Receive Compressed or Decompressed Data message from a socket and launch func_processData() on the extracted data.

Parameters:

isock A descriptor referencing the socket.
channelList Channel list.
network_code Network code. It can be NULL.

Return values:

Pointer to the structure NMXP_DATA_PROCESS on success
NULL on error

int nmxp_sendConnectRequest (int *isock*, char * *naqs_username*, char * *naqs_password*, int32_t *connection_time*)

Sends the message "ConnectRequest" on a socket.

Parameters:

isock A descriptor referencing the socket.
naqs_username User name (maximum 11 characters), zero terminated.
naqs_password Password.
connection_time Time that the connection was opened.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_readConnectionTime (int *isock*, int32_t * *connection_time*)

Read connection time from a socket.

Parameters:

isock A descriptor referencing the socket.
connection_time Time in epoch.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_waitReady (int *isock*)

Wait the message "Ready" from a socket.

Parameters:

isock A descriptor referencing the socket.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

int nmxp_sendDataRequest (int *isock*, int32_t *key*, int32_t *start_time*, int32_t *end_time*)

Sends the message "DataRequest" on a socket.

Parameters:

isock A descriptor referencing the socket.
key Channel key for which data are requested.
start_time Start time of the interval for which data are requested. Epoch time.
end_time End time of the interval for which data are requested. Epoch time.

Return values:

SOCKET_OK on success
SOCKET_ERROR on error

NMXP_CHAN_LIST* **nmxp_getAvailableChannelList (char * *hostname*, int *portnum*,** NMXP_DATATYPE ***datatype*)**

Get the list of available channels from a server.

Parameters:

hostname host name
portnum port number
datatype Type of data contained in the channel.

Returns:

Channel list. It will need to be freed.

Warning:

Returned value will need to be freed.

NMXP_META_CHAN_LIST* nmxp_getMetaChannelList (char * *hostname*, int *portnum*, NMXP_DATATYPE *datatype*, int *flag_request_channelinfo*)

Get the list of the start and end time for the available data for each channel.

Parameters:

hostname host name
portnum port number
datatype Type of data contained in the channel.
flag_request_channelinfo Request information about Network.

Returns:

Channel list. It will need to be freed.

Warning:

Returned value will need to be freed.

int nmxp_raw_stream_seq_no_compare (const void * *a*, const void * *b*)

Base function for qsort() in order to sort an array of pointers to pointers to NMXP_DATA_PROCESS.

Parameters:

a pointer to a pointer to NMXP_DATA_PROCESS
b pointer to a pointer to NMXP_DATA_PROCESS

void nmxp_raw_stream_init (NMXP_RAW_STREAM_DATA * *raw_stream_buffer*, int32_t *max_tollerable_latency*)

Allocate and initialize fields inside a NMXP_RAW_STREAM_DATA structure.

Parameters:

raw_stream_buffer pointer to NMXP_RAW_STREAM_DATA struct to initialize
max_tollerable_latency max tollerable latency in seconds.

void nmxp_raw_stream_free (NMXP_RAW_STREAM_DATA * *raw_stream_buffer*)

Free fields inside a NMXP_RAW_STREAM_DATA structure.

Parameters:

raw_stream_buffer pointer to NMXP_RAW_STREAM_DATA struct to initialize

int nmxp_raw_stream_manage (NMXP_RAW_STREAM_DATA * *p*, NMXP_DATA_PROCESS * *a_pd*, int(p_func_pd*[*NMXP_MAX_FUNC_PD*])(NMXP_DATA_PROCESS *), int *n_func_pd*)**

Execute a list of functions on an chronological ordered array of NMXP_DATA_PROCESS structures.

Parameters:

p pointer to NMXP_RAW_STREAM_DATA
a_pd pointer to NMXP_DATA_PROCESS struct to insert into the array
p_func_pd array of functions to execute on a single item NMXP_DATA_PROCESS
n_func_pd number of functions into the array p_func_pd

int nmxp_raw_stream_manage_flush(NMXP_RAW_STREAM_DATA * *p*, int(p_func_pd*[*NMXP_MAX_FUNC_PD*])(NMXP_DATA_PROCESS *), int *n_func_pd*)**

Execute a list of functions on remaining NMXP_DATA_PROCESS structures.

Parameters:

p pointer to NMXP_RAW_STREAM_DATA
p_func_pd array of functions to execute on a single item NMXP_DATA_PROCESS
n_func_pd number of functions into the array p_func_pd

Le funzioni a cui prestare maggiore attenzione sono quelle che si occupano della gestione del *buffer* dei pacchetti nelle connessioni di tipo *Raw Stream*, cioè dei pacchetti compressi e con valore di *Short-term-complete* uguale a *-1*. Per un canale sismico la funzione *nmxp_raw_stream_manage()* si occupa di riordinare cronologicamente le strutture NMXP_DATA_PROCESS che ad ogni chiamata le vengono passate, successivamente di eseguire sulle stesse le *n_func_pd* funzioni i cui puntatori sono contenuti nell'array *p_func_pd*. Nel caso in cui rilevi una discontinuità temporale del dato, la funzione accoda in un *buffer* la struttura corrente inducendo così una latenza sul flusso dei dati per quel canale. L'attesa dei pacchetti mancanti termina quando il tempo massimo di latenza tollerabile, impostato al momento dell'inizializzazione per mezzo della funzione *nmxp_raw_stream_init()*, viene superato. In quest'ultimo caso la funzione forzerà l'esecuzione delle funzioni sulla prima struttura disponibile causando quindi un gap sul flusso dei dati.

2.3 Uso delle API per sviluppare una nuova applicazione

Per sviluppare una propria applicazione in C che faccia uso della libreria *libnmxp* vengono di seguito illustrati i sorgenti 1 e 2 che possono essere utilizzati come base per l'implementazione dei protocolli *Data Access Protocol 1.0* e *Private Data Stream 1.4*. Su tali strutture di codice C è basato anche *nmxptool* descritto successivamente.

È importante notare come risulti relativamente semplice sviluppare una propria applicazione anche nel caso in cui si vogliano stabilire connessioni di tipo *Raw Stream*. Infatti lo sviluppatore non dovrà far altro che utilizzare la struttura base del sorgente 2, eseguire le opportune personalizzazioni, e dichiarare una funzione con prototipo

int (**process_data_function*) (NMXP_DATA_PROCESS *)

il cui puntatore dovrà poi essere aggiunto nell'array da passare come parametro alla funzione *nmxp_raw_stream_manage()*.

Prima di poter richiamare la funzione *nmxp_raw_stream_manage()* bisogna inizializzare per ogni canale, tramite la funzione *nmxp_raw_stream_init()*, una struttura dati di tipo NMXP_RAW_STREAM_DATA e il valore della massima latenza tollerabile.

Al termine del programma, o comunque al termine della connessione, sarà necessario liberare la memoria allocata dalla struttura NMXP_RAW_STREAM_DATA per mezzo della funzione *nmxp_raw_stream_free()*. Opzionalmente, prima di questa funzione può essere richiamata *nmxp_raw_manage_stream_flush()* che esegue le funzioni sui pacchetti rimanenti indipendentemente dalla continuità del dato.

Sorgente 1. *Struttura base in C che implementa D.A.P. versione 1.0 utilizzando le API di* ***libnmxp***.

```
/* ************************************************************ */
/* Start subscription protocol "DATA ACCESS PROTOCOL" version 1.0 */
/* ************************************************************ */

/* DAP Step 1: Open a socket */
if( (naqssock = nmxp_openSocket(params.hostname, params.portnumberdap)) == NMXP_SOCKET_ERROR) {
  nmxp_log(1, 0, "Error opening socket!\n");
  return 1;
}

/* DAP Step 2: Read connection time */
if(nmxp_readConnectionTime(naqssock, &connection_time) != NMXP_SOCKET_OK) {
  nmxp_log(1, 0, "Error reading connection time from server!\n");
  return 1;
}

/* DAP Step 3: Send a ConnectRequest */
if(nmxp_sendConnectRequest(naqssock, params.datas_username, params.datas_password, connection_time)
!= NMXP_SOCKET_OK) {
  nmxp_log(1, 0, "Error sending connect request!\n");
  return 1;
}

/* DAP Step 4: Wait for a Ready message */
if(nmxp_waitReady(naqssock) != NMXP_SOCKET_OK) {
  nmxp_log(1, 0, "Error waiting Ready message!\n");
  return 1;
}

while(exitdapcondition) {

  /* Start loop for sending requests */
  while(request_SOCKET_OK == NMXP_SOCKET_OK  &&  i_chan < channelList_subset->number) {

    /* DAP Step 5: Send Data Request */
    request_SOCKET_OK = nmxp_sendDataRequest(naqssock, channelList_subset->channel[i_chan].key,
        (double) params.start_time, (double) params.end_time);

    if(request_SOCKET_OK == NMXP_SOCKET_OK) {

      /* DAP Step 6: Receive Data until receiving a Ready message */
      ret = nmxp_receiveMessage(naqssock, &type, &buffer, &length);
      nmxp_log(0, 1, "ret = %d, type = %d\n", ret, type);

      while(ret == NMXP_SOCKET_OK   &&    type != NMXP_MSG_READY) {

        /* Process a packet and return value in NMXP_DATA_PROCESS structure */
        pd = nmxp_processCompressedData(buffer, length, channelList_subset, CURRENT_NETWORK);
        nmxp_data_trim(pd, params.start_time, params.end_time, 0);

        /* Log contents of last packet */
        nmxp_data_log(pd);

        if(pd->buffer) {
          free(pd->buffer);
          pd->buffer = NULL;
        }

        /* Receive Data */
        ret = nmxp_receiveMessage(naqssock, &type, &buffer, &length);
      }

    }
    i_chan++;
  }
  /* DAP Step 7: Repeat steps 5 and 6 for each data request */
} /* END while(exitdapcondition) */

/* DAP Step 8: Send a Terminate message (optional) */
nmxp_sendTerminateSubscription(naqssock, NMXP_SHUTDOWN_NORMAL, "Bye!");

/* DAP Step 9: Close the socket */
nmxp_closeSocket(naqssock);

/* ************************************************************ */
/* End subscription protocol "DATA ACCESS PROTOCOL" version 1.0 */
/* ************************************************************ */
```

Sorgente 2. *Struttura base in C che implementa P.D.S. versione 1.4 utilizzando le API di* ***libnmxp***.

```
int n_func_pd = 0;
int (*p_func_pd[NMXP_MAX_FUNC_PD]) (NMXP_DATA_PROCESS *);

if(params.stc == -1) {

  if(params.flag_logdata) {
    p_func_pd[n_func_pd++] = nmxptool_print_seq_no;
  }

  /* Write Mini-SEED record */
  if(params.flag_writeseed) {
    p_func_pd[n_func_pd++] = nmxptool_write_miniseed;
  }

  /* Send data to SeedLink Server */
  if(params.flag_slink) {
    p_func_pd[n_func_pd++] = nmxptool_send_raw_depoch;
  }
}

/* ********************************************************** */
/* Start subscription protocol "PRIVATE DATA STREAM" version 1.4 */
/* ********************************************************** */

/* PDS Step 1: Open a socket */
if((naqssock = nmxp_openSocket(params.hostname, params.portnumberpds)) == NMXP_SOCKET_ERROR) {
  return 1;
}

/* PDS Step 2: Send a Connect */
if(nmxp_sendConnect(naqssock) != NMXP_SOCKET_OK) {
  printf("Error on sendConnect()\n");  return 1;
}

/* PDS Step 3: Receive ChannelList */
if(nmxp_receiveChannelList(naqssock, &channelList) != NMXP_SOCKET_OK) {
  printf("Error on receiveChannelList()\n");  return 1;
}

/* Get a subset of channel from arguments */
channelList_subset = nmxp_chan_subset(channelList, NMXP_DATA_TIMESERIES, params.channels);

/* PDS Step 4: Send a Request Pending (optional) */

/* PDS Step 5: Send AddChannels */
/* Request Data */
nmxp_sendAddTimeSeriesChannel(naqssock, channelList_subset, params.stc, params.rate,
    (params.flag_buffered)? NMXP_BUFFER_YES : NMXP_BUFFER_NO);

/* PDS Step 6: Repeat until finished: receive and handle packets */
while(exitpdscondition) {
  /* Process Compressed or Decompressed Data */
  pd = nmxp_receiveData(naqssock, channelList_subset, CURRENT_NETWORK);

  /* Log contents of last packet */
  nmxp_data_log(pd);

  /* Manage Raw Stream */
  if(params.stc == -1) {
    nmxp_raw_stream_manage(&(raw_stream_buffer), pd, p_func_pd, n_func_pd);
  }

  if(pd->buffer) {
    free(pd->buffer);  pd->buffer = NULL;
  }
}

/* PDS Step 7: Send Terminate Subscription */
nmxp_sendTerminateSubscription(naqssock, NMXP_SHUTDOWN_NORMAL, "Good Bye!");

/* PDS Step 8: Close the socket */
nmxp_closeSocket(naqssock);

/* ********************************************************** */
/* End subscription protocol "PRIVATE DATA STREAM" version 1.4 */
/* ********************************************************** */
```

3. Il programma nmxptool

Al fine di capire cosa *nmxptool* permette di fare, lanciamo inizialmente il comando che stampa a video terminale l'*help* delle opzioni del comando:

```
kyuzo:~ mtheo$ nmxptool -h
```

```
nmxptool 1.1.2, Nanometrics tool based on libnmxp-1.1.2
        (Data Access Protocol 1.0, Private Data Stream 1.4)
         Support for: libmseed YES, SeedLink YES, Earthworm YES.

Usage: nmxptool -H hostname --listchannels [...]
             Receive list of available channels on the host

       nmxptool -H hostname -C channellist -s DATE -e DATE [...]
             Receive data from hostname by DAP

       nmxptool -H hostname -C channellist [...]
             Receive data from hostname by PDS

       nmxptool nmxptool.d
             Run as earthworm module receiving data by PDS

Arguments:
  -H, --hostname=HOST     Nanometrics hostname.
  -C, --channels=LIST     Channel list STA1.HH?,STA2.??Z,...

Other arguments:
  -P, --portpds=PORT      NaqsServer port number (default 28000).
  -D, --portdap=PORT      DataServer port number (default 28002).
  -N, --network=NET       Declare Network code for all stations (default 'XX').
  -L, --location=LOC      Location code for writing file.
  -v, --verbose           Be verbose.
  -g, --logdata           Print info about data.
  -l, --listchannels      Output list of channel available on NaqsServer.
  -i, --channelinfo       Output list of channel available on NaqsServer and channelinfo.
  -m, --writeseed         Pack received data in Mini-SEED records and write to a file.
  -w, --writefile         Dump received data to a file.
  -k, --slink=plug_name   Send received data to SeedLink like as plug-in.
                          plug_name is set by SeisComP daemon.
                          THIS OPTION MUST BE THE LAST WITHOUT plug_name IN seedlink.ini!
  -V, --version           Print tool version.
  -h, --help              Print this help.

DAP Arguments:
  -s, --start_time=DATE   Start time in date format.
  -e, --end_time=DATE     End time in date format.
                          DATE can be in formats:
                              <date>,<time> | <date>
                          where:
                              <date> = yyyy/mm/dd | yyy.jjj
                              <time> = hh:mm:ss | hh:mm
  -d, --delay=SECs        Receive continuosly data with delay [60..86400].
  -u, --username=USER     DataServer username.
  -p, --password=PASS     DataServer password.

PDS arguments:
  -S, --stc=SECs          Short-term-completion (default -1).
                          -1 is for Raw Stream, no short-term completion.
                           0 chronological order without waiting for missing data.
                          [0..300] wait a period for the gap to be filled by retransmitted packets.
                          Raw Stream is usable only with --rate=-1.
  -R, --rate=Hz           Receive data with specified sample rate (default -1).
                          -1 is for original sample rate and compressed data.
                           0 is for original sample rate and decompressed data.
                          >0 is for specified sample rate and decompressed data.
  -b, --buffered          Request also recent packets into the past.
  -M, --maxlatency=SECs   Max tolerable latency (default 600) [60..600].
                          Usable only with Raw Stream --stc=-1.

Matteo Quintiliani - Istituto Nazionale di Geofisica e Vulcanologia - Italy
Mail bug reports and suggestions to <quintiliani@ingv.it>.
```

Da tale output deduciamo che un parametro sempre necessario è il nome o l'IP del *server* al quale richiedere i dati. Il programma, in funzione dei parametri passati, determina automaticamente se effettuare una connessione al NaqsServer (porta 28000) oppure al DataServer (porta 28002). Se le porte dei *server* non sono quelle di *default* è necessario utilizzare le opzioni ***–P*** e ***–D***. Un primo utilizzo di *nmxptool* per esempio potrebbe essere quello di impiegarlo per reperire la lista dei canali disponibili sul *server* e dei tempo di inizio e fine dei dati per ogni canale. Ciò si ottiene per mezzo del comando:

```
kyuzo:~ mtheo$ nmxptool –H hostname -l
```

Una parte di un possibile output:

```
...
1255538946 USI.HHE.IV      (2007.233,10:39:21.0000  -  2007.243,09:59:44.0000)
1255538945 USI.HHN.IV      (2007.233,16:20:53.0000  -  2007.243,09:59:45.0000)
1255538944 USI.HHZ.IV      (2007.233,22:26:08.0000  -  2007.243,09:59:31.0000)
1238565122 VAGA.HHE.IV     (2007.225,07:10:14.0000  -  2007.243,09:59:19.0000)
1238565121 VAGA.HHN.IV     (2007.225,08:35:24.0000  -  2007.243,09:59:29.0000)
1238565120 VAGA.HHZ.IV     (2007.225,00:03:14.0000  -  2007.243,09:59:29.0000)
...
```

Per ogni canale disponibile viene visualizzato:

- l'indice numerico Nanometrics del canale, denominato *key channel*
- il nome del canale nella forma *Station.Channel.Network*
- data e ora di inizio dei dati disponibili
- data e ora di fine dei dati disponibili

Successivamente potremmo richiedere al DataServer i dati appartenenti ad un certo intervallo di tempo e di un insieme di canali, il comando allora dovrà contenere le opzioni ***-s***, ***-e***, ***-C***, quindi ad esempio:

```
kyuzo:~ mtheo$ nmxptool –H hostname –s 2007.242,00:00 –e 2007/08/30,00:00:05 \
                                                    –C USI.???,VAGA.HHZ -g
```

Osserviamo che la data può essere scritta seguendo tali regole:

DATA,**ORA** oppure solamente **DATA,** dove:

DATA può essere espressa nei seguenti formati:
aaaa/*mm*/*gg*
aaa.*jjj* (*jjj* è il giorno giuliano dell'anno)

ORA può essere espressa nei seguenti formati:
hh:*mm*:*ss*
hh:*mm*

Se si specifica solo **DATA, ORA** verrà automaticamente impostata a 00:00

Notiamo inoltre che la lista dei canali può contenere il carattere speciale ***?*** che ha il significato di "*qualsiasi carattere*". Alla riga di comando abbiamo aggiunto anche l'opzione ***-g*** che visualizza informazioni su ogni pacchetto ricevuto. Ecco un output possibile:

```
XX.USI.HHE 100Hz (2007.242,00:00:00.0000 - 2007.242,00:00:00.8699) lat 130115.1s [1, 48353370] (0)   87pts (-1128, -1128, 1742, 3226, 1) 276
XX.USI.HHE 100Hz (2007.242,00:00:00.8699 - 2007.242,00:00:01.9899) lat 130114.0s [1, 48353371] (0)  112pts (3226, 3226, 2423, 2688, 1) 276
XX.USI.HHE 100Hz (2007.242,00:00:01.9900 - 2007.242,00:00:03.1099) lat 130112.9s [1, 48353372] (0)  112pts (2688, 2688, -548, -686, 1) 276
XX.USI.HHE 100Hz (2007.242,00:00:03.1099 - 2007.242,00:00:04.2500) lat 130111.8s [1, 48353373] (0)  114pts (-686, -686, -857, -74, 1) 276
XX.USI.HHE 100Hz (2007.242,00:00:04.2500 - 2007.242,00:00:05.0000) lat 130111.0s [1, 48353374] (0)   75pts (-74, -74, 1290, 1338, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:00.0000 - 2007.242,00:00:00.2500) lat 130116.8s [1, 49688091] (0)   25pts (301, 301, 11, -143, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:00.2500 - 2007.242,00:00:01.3699) lat 130115.6s [1, 49688092] (0)  112pts (-143, -143, 926, 1534, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:01.3699 - 2007.242,00:00:02.5099) lat 130114.5s [1, 49688093] (0)  114pts (1534, 1534, -220, -17, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:02.5099 - 2007.242,00:00:03.6299) lat 130113.4s [1, 49688094] (0)  112pts (-17, -17, -866, -837, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:03.6300 - 2007.242,00:00:04.7900) lat 130112.2s [1, 49688095] (0)  116pts (-837, -837, -716, -527, 1) 276
XX.USI.HHN 100Hz (2007.242,00:00:04.7899 - 2007.242,00:00:05.0000) lat 130112.0s [1, 49688096] (0)   21pts (-527, -527, 999, 790, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:00.0000 - 2007.242,00:00:00.4400) lat 130116.6s [1, 50549101] (0)   44pts (-5470, -5470, -4031, -4326, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:00.4400 - 2007.242,00:00:01.5599) lat 130115.4s [1, 50549102] (0)  112pts (-4326, -4326, -6154, -6408, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:01.5599 - 2007.242,00:00:02.6799) lat 130114.3s [1, 50549103] (0)  112pts (-6408, -6408, -5355, -5326, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:02.6800 - 2007.242,00:00:03.7999) lat 130113.2s [1, 50549104] (0)  112pts (-5326, -5326, -4203, -4963, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:03.7999 - 2007.242,00:00:04.9199) lat 130112.1s [1, 50549105] (0)  112pts (-4963, -4963, -4980, -5066, 1) 276
XX.USI.HHZ 100Hz (2007.242,00:00:04.9200 - 2007.242,00:00:05.0000) lat 130112.0s [1, 50549106] (0)    8pts (-5066, -5066, -4823, -4804, 1) 276
XX.VAGA.HHZ 100Hz (2007.242,00:00:00.0000 - 2007.242,00:00:00.2999) lat 130116.7s [1, 7848381] (0)   30pts (-10567, -10567, -10553, -10550, 1) 276
XX.VAGA.HHZ 100Hz (2007.242,00:00:00.2999 - 2007.242,00:00:02.5399) lat 130114.5s [1, 7848382] (0)  224pts (-10550, -10550, -10456, -10458, 1) 276
XX.VAGA.HHZ 100Hz (2007.242,00:00:02.5399 - 2007.242,00:00:04.7799) lat 130112.2s [1, 7848383] (0)  224pts (-10458, -10458, -10363, -10362, 1) 276
XX.VAGA.HHZ 100Hz (2007.242,00:00:04.7799 - 2007.242,00:00:05.0000) lat 130112.0s [1, 7848384] (0)   22pts (-10362, -10362, -10331, -10337, 1) 276
```

Per ogni pacchetto ricevuto viene visualizzato:

- il nome del canale nella forma *Network.Station.Channel*
- la frequenza di campionamento
- i tempi del primo e dell'ultimo campione
- la latenza in secondi rispetto all'ora del *client*
- il tipo del pacchetto Nanometrics e il suo numero di sequenza
- il valore del numero di sequenza del più vecchio pacchetto disponibile
- il numero di campioni presenti nel pacchetto
- il valore x_0 contenuto nell'intestazione (*header*) del pacchetto Nanometrics
- il primo e l'ultimo valore della serie di campioni
- il valore x_n, ovvero il valore calcolato che dovrà avere x_0 nel pacchetto successivo
- il flag che indica se x_0 e x_n sono significativi (*0* significativo, *-1* non significativo)
- la lunghezza in *byte* del pacchetto Nanometrics ricevuto

Nell'esempio precedente, non essendo stata definita la rete (*network*), per *default* il programma l'ha impostata a 'XX'. Nel caso avessimo voluto salvare i dati in formato mini-SEED sarebbe stato sufficiente aggiungere l'opzione ***-m*** e il programma avrebbe generato un file per ogni canale. Inoltre, se il DataServer avesse richiesto l'autenticazione si sarebbero dovute utilizzare le opzioni per la definizione del nome utente e della password, ovvero ***-u*** e ***-p***.

Per avere un flusso di dati continuo ma in differita con uno specifico tempo stabilito è possibile utilizzare l'opzione ***-d***. In questo modo si ricevono quindi dati in flusso continuo dal DataServer tenendo fissa la latenza al valore impostato. Ad esempio, per 1 ora (3600 secondi) di differita, un possibile comando sarà:

```
kyuzo:~ mtheo$ nmxptool –H hostname –d 3600 –C USI.???,VAGA.HHZ -g
```

Per ricevere dati in tempo reale, ovvero da un NaqsServer, è sufficiente, in generale, non definire l'intervallo temporale. Quindi un comando del tipo:

```
kyuzo:~ mtheo$ nmxptool –H hostname -C USI.??? –g –R 100
```

restituirebbe un output simile a questo di seguito:

```
XX.USI.HHN 100Hz (2007.243,12:22:48.0000 - 2007.243,12:22:49.0000) lat 9.0s [4, -1] (-1)  100pts (-1, 2080, 2488, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:48.0000 - 2007.243,12:22:49.0000) lat 9.0s [4, -1] (-1)  100pts (-1, 703, 2789, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:49.0000 - 2007.243,12:22:50.0000) lat 8.0s [4, -1] (-1)  100pts (-1, 2947, -1268, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:49.0000 - 2007.243,12:22:50.0000) lat 8.0s [4, -1] (-1)  100pts (-1, 1924, 204, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:49.0000 - 2007.243,12:22:50.0000) lat 8.0s [4, -1] (-1)  100pts (-1, 2490, -1004, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:50.0000 - 2007.243,12:22:51.0000) lat 7.0s [4, -1] (-1)  100pts (-1, -931, 1006, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:50.0000 - 2007.243,12:22:51.0000) lat 7.0s [4, -1] (-1)  100pts (-1, -1131, 1239, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:50.0000 - 2007.243,12:22:51.0000) lat 7.0s [4, -1] (-1)  100pts (-1, -103, -588, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:51.0000 - 2007.243,12:22:52.0000) lat 6.0s [4, -1] (-1)  100pts (-1, 951, 3495, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:51.0000 - 2007.243,12:22:52.0000) lat 6.0s [4, -1] (-1)  100pts (-1, 1318, 790, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:51.0000 - 2007.243,12:22:52.0000) lat 6.0s [4, -1] (-1)  100pts (-1, -467, 93, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:52.0000 - 2007.243,12:22:53.0000) lat 5.0s [4, -1] (-1)  100pts (-1, 365, 956, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:52.0000 - 2007.243,12:22:53.0000) lat 5.0s [4, -1] (-1)  100pts (-1, 3356, 2437, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:52.0000 - 2007.243,12:22:53.0000) lat 5.0s [4, -1] (-1)  100pts (-1, 1034, 1527, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:53.0000 - 2007.243,12:22:54.0000) lat 4.0s [4, -1] (-1)  100pts (-1, 951, 16, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:53.0000 - 2007.243,12:22:54.0000) lat 4.0s [4, -1] (-1)  100pts (-1, 2559, -319, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:53.0000 - 2007.243,12:22:54.0000) lat 4.0s [4, -1] (-1)  100pts (-1, 1472, 675, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:54.0000 - 2007.243,12:22:55.0000) lat 3.0s [4, -1] (-1)  100pts (-1, 255, -351, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:54.0000 - 2007.243,12:22:55.0000) lat 3.0s [4, -1] (-1)  100pts (-1, -668, 1457, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:54.0000 - 2007.243,12:22:55.0000) lat 3.0s [4, -1] (-1)  100pts (-1, 1101, 1541, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:55.0000 - 2007.243,12:22:56.0000) lat 2.0s [4, -1] (-1)  100pts (-1, -540, 1162, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:55.0000 - 2007.243,12:22:56.0000) lat 2.0s [4, -1] (-1)  100pts (-1, 1593, -488, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:55.0000 - 2007.243,12:22:56.0000) lat 2.0s [4, -1] (-1)  100pts (-1, 1608, 1355, -1, 0) 420
XX.USI.HHE 100Hz (2007.243,12:22:56.0000 - 2007.243,12:22:57.0000) lat 1.0s [4, -1] (-1)  100pts (-1, 1324, -1674, -1, 0) 420
XX.USI.HHN 100Hz (2007.243,12:22:56.0000 - 2007.243,12:22:57.0000) lat 2.0s [4, -1] (-1)  100pts (-1, -371, 2315, -1, 0) 420
XX.USI.HHZ 100Hz (2007.243,12:22:56.0000 - 2007.243,12:22:57.0000) lat 2.0s [4, -1] (-1)  100pts (-1, 1279, 967, -1, 0) 420
```

L'opzione ***-R*** è stata utilizzata per dichiarare che i pacchetti da ricevere sarebbero stati scompattati dal server con una frequenza di 100Hz. Notiamo infatti che il pacchetto è di tipo 4, ovvero decompresso, con una capacità fissa di un secondo e che x_0 e x_n non sono significativi. Invece per i pacchetti compressi (pacchetto di tipo 1) avremmo anche pouto specificare, tramite l'opzione ***-S***, un valore fra *1* e *300* secondi dello *Short-term-complete*, oppure *0* per *nessun Short-term-complete*, oppure uguale *-1* per ricevere i pacchetti in modalità *Raw Stream*.

Quest'ultimo caso rappresenta una delle funzionalità più importanti di *nmxptool* poiché consente di ricevere i pacchetti in modo continuo in tempo reale, in ordine cronologico, con minima latenza e minimo numero di *gap*. Il programma è in grado di gestire il *buffering* dei pacchetti trasmessi e ritrasmessi, il loro

riordinamento e l'esecuzione delle operazione selezionate tramite le opzioni. Un'opzione collegata a questa gestione è ***-M*** che serve a specificare la massima latenza tollerabile nell'attesa di un pacchetto mancante. Di conseguenza da tale opzione dipende la grandezza del *buffer*.

Quando si interagisce con il NaqsServer si può anche utilizzare l'opzione ***-b***, la quale permette di ricevere anche alcuni dati, in quantità discrezionale del server, che precedono quelli dell'istante attuale di richiesta.

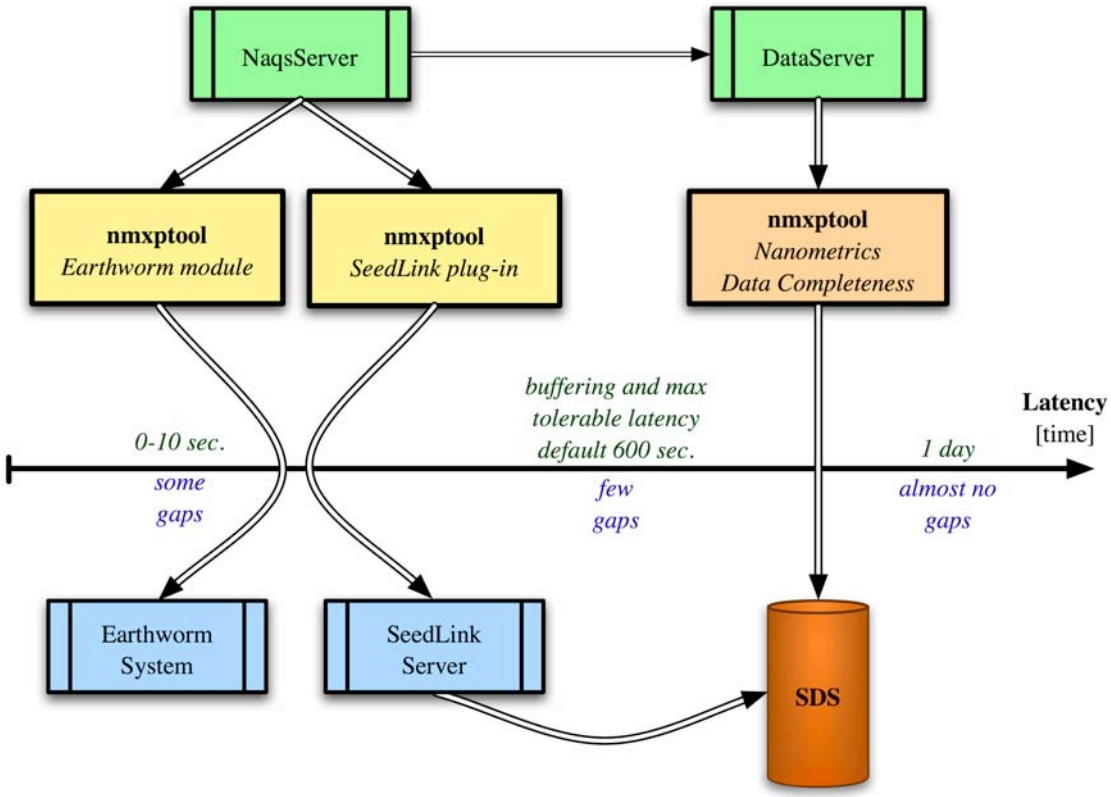


Figura 3. *Localizzazione degli eventi e archiviazione dei dati sismici in tempo reale e completamento in differita. Le tre attività si basano con modalità diverse su nmxptool. Nel primo e nel secondo caso, nmxptool si connette al NaqsServer in modalità Raw Stream e viene eseguito rispettivamente come modulo del sistema Earthworm e come plug-in SeedLink. Nel terzo i dati mancanti vengono richiesti da nmxptool al DataServer e ricongiunti alla struttura di archiviazione SDS di SeisComP.*

3.1 Modulo Earthworm

nmxptool può essere eseguito come modulo del sistema ***Earthworm***. Generalmente il tipo di connessione eseguita è di tipo *Raw Stream* e i parametri, invece di essere passati tramite linea di comando, vengono letti da un file di configurazione tipo *.d*, rispettando così lo standard dei moduli *Earthworm*. All'interno della distribuzione sono disponibili i due file *nmxptool.d* e *nmxptool.desc*, i quali possono essere usati come base per la configurazione di *nmxptool* all'interno del sistema *Earthworm*. È comunque in corso la richiesta per inserire *nmxptool* nelle distribuzioni ufficiali di ***Earthworm***.

3.2 Plug-in SeedLink

Con qualsiasi configurazione di opzioni descritte precedentemente, *nmxptool* può essere lanciato come un plug-in per ***SeedLink*** per mezzo dell'utilizzo dell'opzione ***-k***. Questa opzione deve essere necessariamente dichiarata per ultima. All'interno della distribuzione sono inoltre disponibili i *templates*

SeedLink necessari alla configurazione del plug-in tramite il comando "*seiscomp config*". È comunque in corso la richiesta per inserire *nmxptool* fra i *plug-in* delle distribuzioni ufficiali di ***SeisComP***.

3.3 Completezza del dato Nanometrics

La figura 3 illustra come *nmxptool* viene utilizzato all'interno dell'INGV per far fluire i dati sismici delle stazioni che trasmettono tramite i protocolli Nanometrics nei sistemi *Earthworm* e *SeisComP*. Le forme d'onda vengono ricevute in tempo reale in modalità *Raw Stream* al fine di minimizzare la latenza e il numero di *gap*. *nmxptool* viene configurato e lanciato all'interno del sistema ***Earthworm*** per consentire il calcolo delle localizzazioni degli eventi sismici ed inoltre viene configurato e lanciato all'interno del sistema ***SeisComP*** come *plug-in SeedLink* per l'archiviazione dei dati. La latenza indotta dal programma è determinata solo nel caso in cui si rimanga in attesa di uno o più pacchetti mancanti. Tale attesa termina nel momento in cui il *buffer* risulti completamente pieno comportando quindi una perdita di dati (*gap*). Il valore impostato per la massima latenza tollerabile sarà, in generale, minore per localizzare un evento (ad esempio 30-60 sec.) rispetto a quello impostato per l'archiviazione (ad esempio 300-600 sec.).

Al fine di garantire completezza dei dati archiviati è stata sviluppata una procedura dal nome *nmdc*, ovvero "Nanometrics Data Completeness", che basandosi sulla versatilità di *nmxptool* recupera i dati mancanti dopo qualche ora o il giorno successivo. In questo caso i *gap* risultanti non potranno più essere colmati poiché i dati richiesti non risultano più essere definitivamente presenti sul lato dei *server* Nanometrics.

4. Conclusioni

Lo sviluppo e i test eseguiti in questi ultimi due mesi, hanno permesso di realizzare una libreria nel suo complesso stabile ed efficiente. Considerando congiuntamente *libnmxp* e *nmxptool*, anche il numero di funzionalità implementate risulta essere molto soddisfacente. La più importante fra tutte è sicuramente la gestione delle connessioni di tipo *Raw Stream*, che riesce a garantire una bassa latenza e nel contempo un numero minino di *gap*. Grazie a questa caratteristica *nmxptool* apporta, per quanto concerne l'acquisizione da *server* Nanometrics, un fondamentale contributo alle comunità degli utilizzatori dei sistemi ***SeisComP*** e ***Earthworm***. Infatti sia *naqs_plugin*, l'attuale plug-in per SeedLink, che *naqs2ew*, l'attuale modulo per Earthworm, non essendo in grado di gestire connessioni di tipo *Raw Stream*, non possono garantire minimamente la continuità del dato al verificarsi di una ritrasmissione dal lato *Nanometrics Server – Stazione Sismica* (fig. 1).

Le tabelle 2 e 3 mostrano i *reports* sintetici dei dati archiviati per alcuni canali in test il 22 e il 23 settembre 2007. Giornalmente, per ogni canale viene visualizzato:

- *Totale di pacchetti ritrasmessi*: per una trasmissione di tipo *short-term-complete* i dati contenuti in questi pacchetti sarebbero stati persi e avrebbero causato dei *gap*.
- *Massima latenza registrata*: la massima latenza registrata durante il giorno e indotta dall'attesa dei pacchetti mancanti. Per tale test la latenza massima tollerabile è stata impostata a 600 secondi.
- *Numero di gap ottenuti in tempo reale* tramite l'acquisizione dei dati per mezzo di *nmxptool* usato come *plug-in* SeedLink. Possiamo notare come il numero di *gap* ottenuti in tempo reale dipenda fortemente dall'attesa dei pacchetti ritrasmessi.
- *Numero di gap definitivi* ottenuti recuperando i dati il giorno dopo dal DataServer per mezzo di *nmxptool* usato all'interno della procedura *nmdc*, "Nanometrics Data Completeness".
- *Percentuale dei pacchetti persi in tempo reale*: il valore è il risultato della seguente espressione: [(Gap Tempo Reale – Gap Definitivi) / Pacchetti Ritrasmessi] * 100. Questo valore può essere interpretato come espressione della bontà nella scelta del valore di massima latenza tollerabile per quel dato canale. Su questo valore e secondariamente sulla latenza massima sono state ordinate le due tabelle.

Normalmente il dato definitivo dovrebbe essere continuo e completo, quindi la presenza di un numero rilevante di *gap* dovrebbe evidenziare in qualche modo un'anomalia o un problema nel sistema di acquisizione. È questo infatti il caso verificatosi per la stazione di MONC durante il test. Esaminando i *log* di *nmxptool* si constata che tali *gap* sono fittizi poiché dovuti a errati valori temporali all'interno dei pacchetti e quindi probabilmente determinati da un mal funzionamento del GPS.

Al momento *nmxptool* viene utilizzato come plug-in SeedLink anche nei seguenti istituti di ricerca ai quali va anche un riconoscimento per la loro collaborazione in fase di test e *debugging* del software:

- *National Data Center,* **Israele.** (Guy Tikochinsky)
- *Institute of Geodynamics, National Observatory of Athens,* **Grecia**. (Nicos Melis)

Possiamo quindi sintetizzare i risultati ottenuti rilevando che l'utilizzo di *nmxptool* con connessioni in tempo reale al NaqsServer (*online*) di tipo *Raw Stream*, garantisce un ottimo compromesso fra continuità del dato e latenza indotta, mentre il suo utilizzo con connessioni in differita al DataServer (*offline*) garantisce pienamente la completezza del dato (fig. 3).

È evidente inoltre che la versalità di *nmxptool* e il numero di funzionalità offerte, sono un valido supporto a procedure che necessitano di dati a richiesta, come ad esempio il calcolo della magnitudo o del momento tensore dopo un evento sismico.

Per il futuro l'autore intende manutenere *libnmxp* e *nmxptool* ed ampliare le funzionalità della libreria anche per quanto riguarda la gestione dei dati di tipo *serial data*, *trigger, event* e *state-of-health*.

Canale	***Pacchetti Ritrasmessi***	***Latenza Massima (sec.)***	***Gap Tempo reale*** *SL plug-in*	***Gap Definitivi*** *nmdc*	***Percentuale Persi in tempo reale***
NOCI.HHZ	2	13.4	0	0	0.0
AMUR.HHZ	2	13.8	0	0	0.0
SALO.HHN	3	16.4	0	0	0.0
SALO.HHE	9	19.1	0	0	0.0
SALO.HHZ	13	19.1	0	0	0.0
NOCI.HHN	5	19.6	0	0	0.0
NOCI.HHE	3	20.1	0	0	0.0
MABI.HHZ	6	26.7	0	0	0.0
AMUR.HHN	6	37.5	0	0	0.0
AMUR.HHE	6	38.5	0	0	0.0
SCTE.HHZ	9	44.2	0	0	0.0
SCTE.HHN	8	45.2	0	0	0.0
SCTE.HHE	11	46.2	0	0	0.0
DOI.HHN	6	46.5	0	0	0.0
MABI.HHN	13	46.8	0	0	0.0
CARO.HHZ	4	47.0	0	0	0.0
CARO.HHN	13	47.3	0	0	0.0
DOI.HHZ	13	47.4	0	0	0.0
MABI.HHE	12	47.6	0	0	0.0
MSAG.HHZ	32	56.0	0	0	0.0
MRGE.HHZ	12	56.1	0	0	0.0
SGRT.HHZ	38	56.1	0	0	0.0
MRGE.HHE	11	56.8	0	0	0.0
MSAG.HHN	29	57.0	0	0	0.0
SGRT.HHN	31	57.0	0	0	0.0
MRGE.HHN	16	57.5	0	0	0.0
MSAG.HHE	34	57.5	0	0	0.0
SGRT.HHE	38	57.9	0	0	0.0
MONC.HHZ	10	67.7	2	2	0.0
TIR.HHN	18	68.7	0	0	0.0
MONC.HHN	16	68.8	3	3	0.0
TIR.HHZ	15	69.4	0	0	0.0
MONC.HHE	20	317.6	39	39	0.0
DOI.HHE	10	317.8	0	0	0.0
TIR.HHE	17	323.3	0	0	0.0
CARO.HHE	12	329.9	0	0	0.0
USI.HHZ	19	588.6	1	0	5.2
USI.HHN	18	589.5	1	0	5.5
USI.HHE	17	590.2	1	0	5.8
BOB.HHN	13	590.0	1	0	7.6
BOB.HHE	9	589.9	1	0	11.1
BOB.HHZ	6	588.8	1	0	16.6
MDI.HHZ	16	601.2	3	0	18.7
MDI.HHE	14	600.9	3	0	21.4
MDI.HHN	17	608.9	7	0	41.1

Tabella 2. *Report sintetico relativo ai dati archiviati dei canali in test il 22 settembre 2007 tramite nmxptool e seedllink.*

Canale	***Pacchetti Ritrasmessi***	***Latenza Massima (sec.)***	***Gap Tempo reale*** *SL plug-in*	***Gap Definitivi*** *nmdc*	***Percentuale Persi in tempo reale***
AMUR.HHN	2	14.4	0	0	0.0
NOCI.HHZ	2	15.3	0	0	0.0
AMUR.HHZ	2	16.3	0	0	0.0
NOCI.HHN	3	17.0	0	0	0.0
AMUR.HHE	2	18.3	0	0	0.0
NOCI.HHE	1	19.2	0	0	0.0
DOI.HHZ	11	24.9	0	0	0.0
DOI.HHE	4	26.5	0	0	0.0
DOI.HHN	5	28.0	0	0	0.0
BOB.HHZ	12	31.2	0	0	0.0
BOB.HHE	12	31.4	0	0	0.0
BOB.HHN	9	31.9	0	0	0.0
SCTE.HHE	1	36.0	0	0	0.0
SCTE.HHZ	4	36.2	0	0	0.0
SCTE.HHN	6	39.1	0	0	0.0
MRGE.HHZ	13	45.6	0	0	0.0
CARO.HHE	11	46.0	0	0	0.0
MSAG.HHZ	29	46.4	0	0	0.0
MABI.HHE	11	46.5	0	0	0.0
MSAG.HHN	21	46.5	0	0	0.0
SALO.HHZ	17	46.9	0	0	0.0
CARO.HHN	7	47.0	0	0	0.0
MSAG.HHE	20	47.0	0	0	0.0
SALO.HHE	12	47.2	0	0	0.0
MABI.HHN	14	47.3	0	0	0.0
MABI.HHZ	6	47.6	0	0	0.0
CARO.HHZ	12	47.8	0	0	0.0
SGRT.HHN	23	56.3	0	0	0.0
MONC.HHZ	11	56.4	3	3	0.0
MONC.HHN	15	56.9	2	2	0.0
MRGE.HHE	14	57.4	0	0	0.0
MRGE.HHN	11	57.4	0	0	0.0
SGRT.HHE	31	57.5	0	0	0.0
SGRT.HHZ	21	57.5	0	0	0.0
TIR.HHE	18	58.8	0	0	0.0
USI.HHZ	23	59.2	0	0	0.0
USI.HHN	13	67.7	0	0	0.0
TIR.HHN	12	324.8	0	0	0.0
MONC.HHE	16	325.1	34	34	0.0
MDI.HHE	10	326.2	0	0	0.0
TIR.HHZ	9	326.3	0	0	0.0
SALO.HHN	14	327.3	0	0	0.0
USI.HHE	20	600.5	1	0	5.0
MDI.HHN	6	603.6	3	0	50.0
MDI.HHZ	4	607.1	2	0	50.0

Tabella 3. *Report sintetico relativo ai dati archiviati dei canali in test il 23 settembre 2007 tramite nmxptool e seedllink.*

5. Ringraziamenti

Un particolare ringraziamento va al Dott. Salvatore Mazza per la fiducia che sempre mi riserva. Sono inoltre riconoscente al Dott. Marco Olivieri per il suo supporto nel controllo di qualità dei dati prodotti dalle mie applicazioni.

6. Bibliografia e riferimenti web

Nanometrics, Inc., (1989-2002), Libra Satellite Seismograph System – Training Course Notes.

SeisComP, The Seismological Communication Processor
http://www.gfz-potsdam.de/geofon/seiscomp/

Earthworm, Seismic network data acquisition and processing system
http://www.isti2.com/ew/

libmseed, 2.1.4, The Mini-SEED library
http://www.iris.edu/manuals/

Doxygen, Source code documentation generator tool
http://www.stack.nl/~dimitri/doxygen/

GNU General Public License
http://www.gnu.org/copyleft/gpl.html