

# Le projet HomeSIP : la domotique avec le protocole SIP

Patrice Kadionik, Maître de Conférences à l'ENSEIRB ([kadionik@enseirb.fr](mailto:kadionik@enseirb.fr))

Mars 2006

## **Introduction**

Cet article présente le projet HomeSIP. Il consiste à mettre en place une plateforme domotique à l'ENSEIRB (*Ecole Nationale Supérieure d'Informatique Electronique et Radiocommunications de Bordeaux*) mettant en oeuvre le protocole SIP. Ce projet est par nature un projet orienté embarqué, composé de différents systèmes électroniques sous Linux embarqué.

Ce projet sera dans un premier temps décrit puis positionné dans le cadre plus général du « M2M » (*Machine to Machine*). Une introduction sur le protocole SIP sera ensuite faite afin de bien cerner l'intérêt d'utiliser un tel protocole. La plateforme matérielle actuelle et son évolution seront ensuite décrites ainsi que les développements logiciels associés mettant en oeuvre Linux.

## **Présentation du projet HomeSIP. Le M2M**

HomeSIP est l'acronyme de « *Home Automation with SIP* ». Plus clairement, il s'agit d'un projet domotique basé sur la mise en oeuvre du protocole SIP (*Session Initiation Protocol*) utilisé généralement pour la téléphonie sur Internet VoIP (*Voice Over IP*).

L'idée est donc d'utiliser SIP pour collecter des informations provenant de différents capteurs et de piloter aussi différents équipements (actionneurs).

Le projet HomeSIP consiste donc :

- En une infrastructure matérielle composée de capteurs et d'actionneurs connectés à des systèmes embarqués qui sont communicants et qui possèdent tous une connectivité IP.
- A mettre en place les logiciels adéquats dans les systèmes embarqués sous Linux.
- A développer des langages dédiés de type DSL (*Domain Specific Language*) pour développer de nouveaux services autour de la plateforme.

Le projet HomeSIP sera intégré au final à la plateforme TelIP (Téléphonie par IP) en exploitation à l'ENSEIRB.

La plateforme TelIP permet des communications VoIP au sein de l'école mais autorise aussi la passerelle vers le réseau téléphonique classique RTC (Réseau Téléphonique Commuté). TelIP offre surtout des services intelligents développés à l'aide d'un nouveau langage dédié de haut niveau. On peut citer comme services l'adaptation des pièces jointes ou d'un flux vidéo en fonction des capacités de son « périphérique SIP » (PC, PDA, téléphone...). Ce travail a été réalisé par l'équipe Phoenix du Labri (Laboratoire Bordelais de Recherche en Informatique).

HomeSIP est donc la continuité de TelIP et l'interopérabilité entre les deux plateformes est totale car toutes deux basées sur le protocole SIP, protocole hautement interopérable.

Le projet HomeSIP est par nature un projet de domotique, la domotique étant en fait une

composante du marché du M2M. Mais qu'est-ce que le M2M ?

Le M2M (*Machine to Machine*) est une infrastructure basée autour d'un réseau qui autorise des communications directement entre équipements ou via un serveur central sans aucune intervention humaine. Cela permet par exemple la capture automatique de données et leur traitement par les équipements après transmission.

Ce marché émergent est en fait l'évolution « naturelle » de la mise en place de la connectivité réseau et en particulier la connectivité IP dans les équipements électroniques.

Le marché du M2M s'appuie aussi sur les mutations technologiques opérées depuis quelques années, conséquence de l'inexorable loi de Moore :

- Les protocoles Internet sont des standards de fait. Ils sont facilement intégrables dans les équipements électroniques.
- Les fonctionnalités électroniques sont toujours de plus en plus intégrées.
- Les matériels électroniques sont puissants et bon marché : capteurs miniatures MEMS (*Micro Electro Mechanical System*), tags RFID (*Radio Frequency ID*)...
- Le processeur 32 bits va devenir dans les années qui viennent le processeur de base dans l'embarqué et va être prépondérant en volume par rapport au processeur 8 ou 16 bits. C'est une condition nécessaire pour pouvoir mettre en oeuvre Linux.
- On dispose maintenant de connexions réseaux permanentes à bas prix qui ne sont plus facturées à la durée : réseau ADSL, réseau GPRS...
- Les connexions sans fil sont de plus en plus utilisées. Le coût d'installation est aussi intéressant car l'on minimise le câblage.
- On dispose de standards, synonymes d'interopérabilité et de maîtrise des coûts : normes RFC pour les protocoles Internet, normes UIT (ADSL...), normes ETSI (GSM, GPRS), normes IEEE (Wifi, Zigbee)...

Le M2M correspond donc à la convergence électronique-informatique communicantes et souvent par Internet. On parle d'Internet diffus ou ambiant (*ubiquitous Internet*) et d'informatique massivement répartie ou informatique diffuse (*ubiquitous computing*).

Selon le cabinet d'étude IDATE (<http://www.idate.fr>) qui a publié en juin 2005 une étude récente sur ce sujet, le marché du M2M se compte en milliards d'équipements électroniques et en centaines de milliards d'objets ou capteurs communicants.

Selon IDATE, 92 millions de modules ont été vendus en 2004, quelle que soit la connectivité réseau choisie. Ce chiffre devrait atteindre 500 millions de modules d'ici 2010 pour 2 milliards de machines et de 100 milliards d'objets communicants.

Le M2M se développe autour d'industries verticales par secteur d'activité. On peut citer comme domaines d'application :

- La gestion de sites et de bâtiments, la domotique.
- La sécurité, la vidéosurveillance, la navigation...
- Les transports, la gestion de flotte automobile.
- La télémesure, le relevé de compteurs.
- La gestion de l'énergie.
- La télémédecine.
- La monétique.
- Le commerce.
- ...

## ***Le protocole SIP : une petite introduction***

### **Structure générale du protocole SIP**

Le protocole SIP (*Session Initiation Protocol*) fait partie de la famille des protocoles Internet. Cela correspond en fait à plusieurs RFC (*Request For Comments*) disponibles en ligne sur <http://rfc-editor.org/> :

- RFC 3261 : *SIP: Session Initiation Protocol*.
- RFC 3265 : *Session Initiation Protocol (SIP)-Specific Event Notification*.
- RFC 3428 : *Session Initiation Protocol (SIP) Extension for Instant Messaging*.

Le groupe de travail SIMPLE (*SIP for Instant Messaging and Presence Leveraging Extensions*) est aussi à considérer (<http://www.ietf.org/html.charters/simple-charter.html>).

SIP est un protocole de signalisation de bout en bout permettant d'établir une session entre deux équipements pour un échange de données (ou d'un flux) par Internet. Il est le descendant des protocoles de signalisation plus classiques comme SS7 (Système de Signalisation numéro 7) de l'UIT-T (Union Internationale des Télécommunications) que l'on retrouve dans le RTC (Réseau Téléphonique Commuté), le RNIS (Réseau Numérique à Intégration de Services), le GSM...

Il s'en distingue radicalement par sa flexibilité et surtout par le fait qu'il n'est pas lié à un type de données à échanger et encore moins à un type de réseau de transport.

Cette qualité est sa grande force. On peut l'utiliser pour mettre en relation 2 équipements pour tout type d'échange de données :

- Voix sur IP. C'est l'échange de données le plus connu.
- Messagerie instantanée IM (*Instant Messaging*).
- Vidéo.
- Autres : indication de présence, notification d'événements...

SIP (dont la première version date de 1997 et la version courante date de juin 2002) possède aussi un concurrent normalisé par l'UIT-T, la norme H.323 qui n'a pas su et pu s'imposer !

Les messages SIP peuvent être transportés par :

- Le protocole de transport UDP. C'est le cas le plus usuel.
- Le protocole TCP.
- Le protocole TLS (*Transport Layer Security*, RFC 3546) utilisant SSL (*Secure Socket Layer*) sur TCP.
- Le protocole SCTP (*Stream Control Transport Protocol*, RFC 2960).

La figure 1 donne le positionnement de SIP dans la constellation des protocoles Internet.

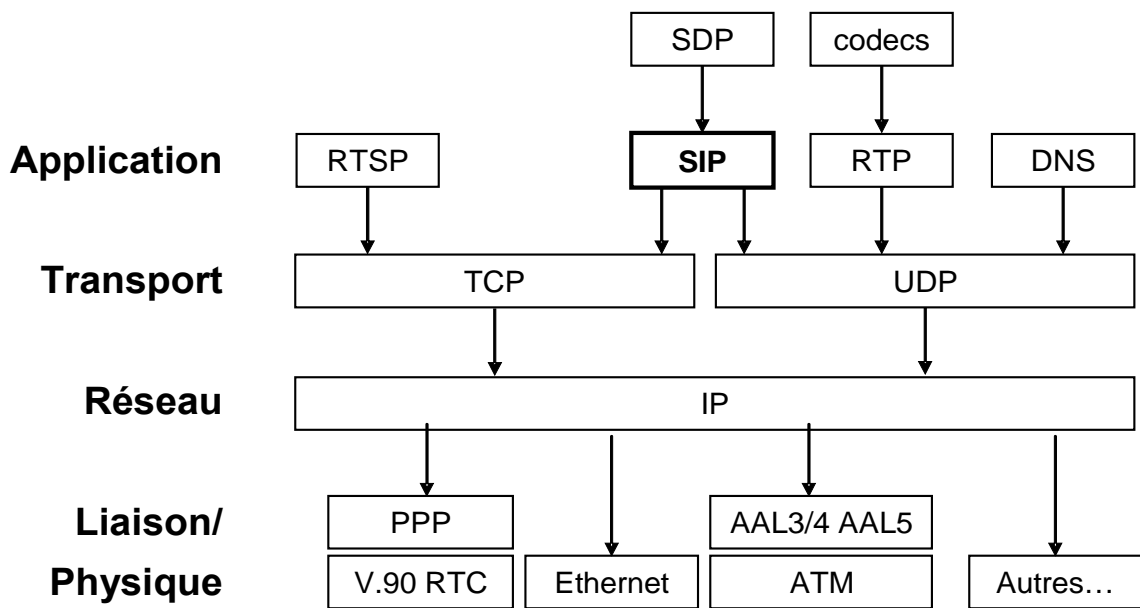


Figure 1 : Positionnement du protocole SIP parmi les protocoles Internet

SIP reprend des éléments techniques des protocoles SMTP (*Simple Mail Transport Protocol*) et HTTP (*Hyper Text Transport Protocol*) ; ce qui veut dire que :

- SIP est basé sur le concept d'application client/serveur. On a une application d'extrémité cliente qui désire se mettre en relation avec une application d'extrémité serveur. SIP possède un numéro de port réservé : le 5060.
- SIP est un protocole de type commande/réponse. Ces commandes/réponses sont structurées sous forme de chaînes de caractères ASCII directement lisibles par l'humain et ressemblent étrangement à celles de HTTP (les codes d'état pour les réponses par exemple)
- SIP possède un adressage pour identifier les applications SIP : c'est l'URI (*Uniform Resource Identifier*) comme on a l'URL (*Uniform Resource Locator*) pour HTTP. Un exemple d'URI est : `sip:kadionik@enseirb.fr`. Cela ne change pas trop d'une URL du type : `http://www.enseirb.fr` !

L'URI dans sa forme la plus générale s'écrit :

- `sip:username@hostname:port;paramètres?header`
- ou `sips:username@hostname:port;paramètres?header` (mode sécurisé)

### Infrastructure SIP

Diverses entités (équipements et/ou applications) sont mises en oeuvre par SIP :

- L'agent utilisateur SIP (*SIP User Agent*). On distingue l'application cliente qui initie un appel

SIP et émet une requête SIP (*Client UA* ou C-UA) et l'application serveur appelée (*Server UA* ou S-UA) qui répond à une requête SIP.

- Le serveur de redirection (*Redirect Server*). Il aide à localiser un agent SIP.
- Le serveur d'enregistrement (*Registrar Server*). Il enregistre la disponibilité d'un agent SIP. Il traite plus particulièrement le message SIP REGISTER. Il peut être intégré à un serveur proxy.
- Le serveur proxy (*Proxy Server*). Il agit comme un proxy HTTP.
- Le serveur de localisation (*Location Server*). Ce n'est pas vraiment une entité mais plutôt une base de données utilisée par un serveur de redirection et/ou un serveur proxy.

La figure 2 donne un exemple d'infrastructure SIP.

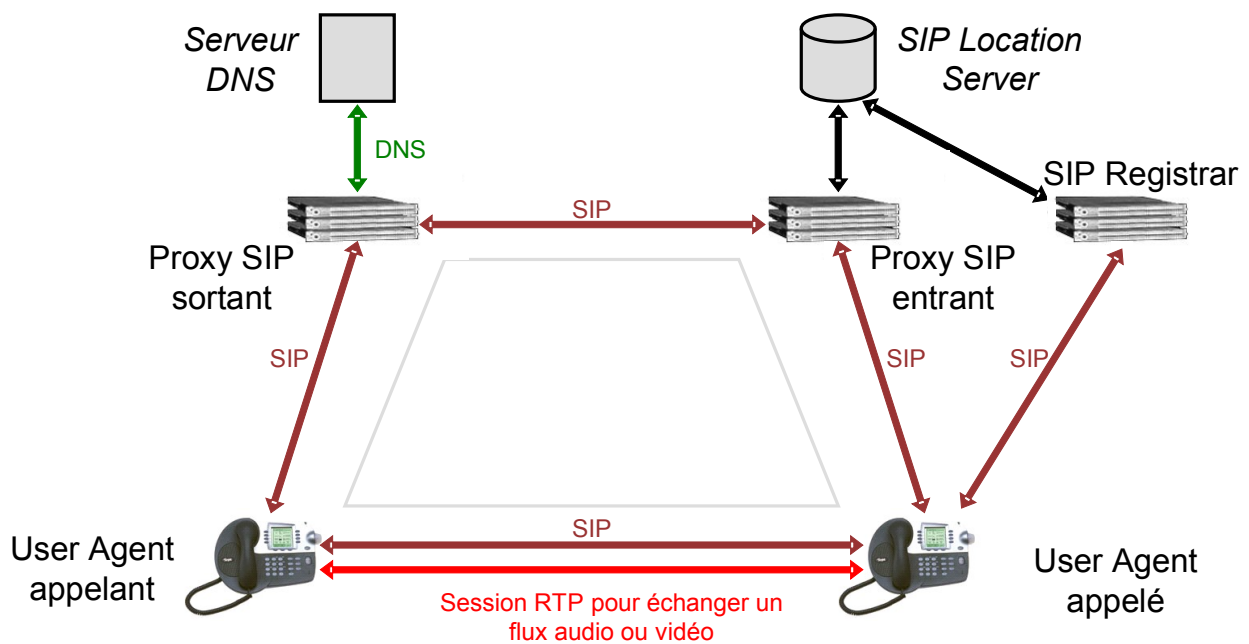


Figure 2 : Infrastructure SIP

## Messages SIP

En notation informatique ABNF (*Augmented Backus Naur Form*), un message SIP est soit une commande, soit une réponse :

SIP-message = Request / Response

Une commande SIP est de la forme :

```
Request = Request-Line
         *( message-header )
         CRLF
         [ message-body ]
```

Request-Line = Method SP Request-URI SP SIP-Version CRLF

On retrouve ni plus ni moins la structure d'une commande HTTP.

Les méthodes autorisées sont :

- REGISTER : message d'enregistrement d'un agent SIP à un *Registrar Server*.
- INVITE : requête SIP pour établir une session entre 2 agents SIP.
- CANCEL : message pour annuler la requête INVITE en cours.
- ACK : acquittement d'une requête INVITE.
- BYE : fin d'une session entre 2 agents SIP.
- SUBSCRIBE, NOTIFY : souscription et notification d'événements.
- MESSAGE : messagerie instantanée.
- Autres : REFER, OPTIONS, INFO.

Une réponse SIP est de la forme :

```
Response          = Status-Line
                   * ( message-header )
                   CRLF
                   [ message-body ]
```

```
Status-Line      = SIP-Version SP Status-Code SP Reason-Phrase CRLF
```

On retrouve ni plus ni moins la structure d'une réponse HTTP. La figure 3 présente un enchaînement typique pour établir une communication VoIP par SIP.

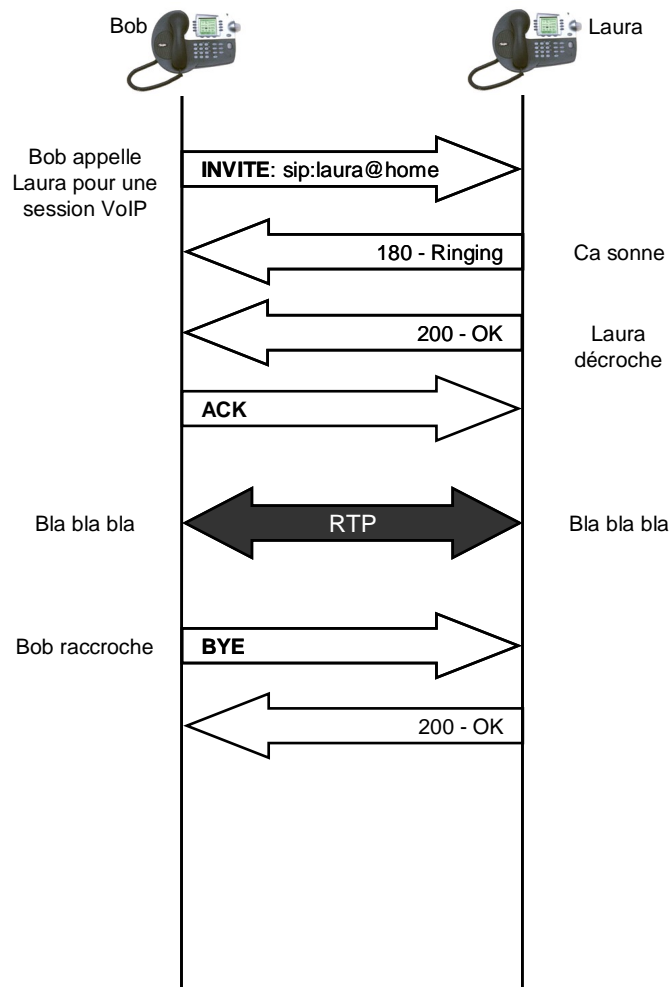


Figure 3 : Echange de messages SIP pour une communication VoIP

Plus précisément, voici un exemple de traces d'une requête SIP INVITE :

```

INVITE sip:laura@home.com SIP/2.0
Via: SIP/2.0/UDP pc11.work.com
Max-Forwards: 70
To: Laura <sip:laura@home.com>
From: Bob <sip:bob@work.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@pc11.work.com>
Content-Type: application/sdp
Content-Length: 131
  
```

```

v=0
o=Bob 289123451 289123451 IN IP4 111.22.33.44
s=Let us talk for a while
c=IN IP4 111.22.33.44
t=0 0
m=audio 20002 RTP/AVP 0
a=rtpmap:0 PCMU/8000
  
```

et d'une réponse SIP :

```

SIP/2.0 200 OK
To: Laura <sip:laura@home.com>;tag=a6c85cf
  
```

```
From: Bob <sip:bob@work.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:laura@222.33.44.55>
Content-Type: application/sdp
Content-Length: 131
```

```
v=0
o=Laura 289123444 289123444 IN IP4 222.33.44.55
s=Let us talk for a while
c=IN IP4 222.33.44.55
t=0 0
m=audio 41002 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Le corps d'une commande/réponse respecte le format MIME (*Multipurpose Internet Mail Extensions*) dont l'en-tête MIME est sur l'exemple :

```
Content-Type: application/sdp
Content-Length: 131
```

et le corps MIME :

```
v=0
o=Laura 289123444 289123444 IN IP4 222.33.44.55
...
```

On respecte dans le cas présent, le format SDP (*Session Description Protocol*) qui sert à décrire la session établie :

- Nom de la session.
- Paramètres des flux audio, vidéo...
- Adresses IP et numéros de port à utiliser.
- ...

Sur l'exemple donné, on établit une session pour l'échange d'un flux audio en mode PCM 8kHz.

## **Le protocole SIP et la domotique**

### **Adéquation du protocole SIP avec la domotique**

Le protocole SIP possède des qualités intrinsèques pour la domotique :

- Il supporte un mode d'adressage abstrait.
- Il apporte un niveau de confidentialité et de sécurité. Les données échangées peuvent être authentifiées et chiffrées.
- Il supporte différents flux d'échange et différents mécanismes de communication.
- Il supporte tout type de charge (*payload*) dans ses messages en utilisant le format MIME.
- Il peut s'interfacer à d'autres technologies domotiques via des passerelles : bus CAN (*Control Area Network*), courant porteur (norme domotique X10)...
- Il supporte la mobilité.
- Il réutilise l'infrastructure SIP classique.

En domotique, on distingue 2 types de transferts de données :

- Le transfert synchrone. Une donnée actuelle est acquise par une application, par exemple la valeur courante d'un capteur. On peut vouloir aussi réaliser une action, par exemple changer l'état courant d'un actionneur. Un capteur peut être un capteur de température, un actionneur, un gâchette électrique de porte. On doit pouvoir réaliser des actions « GET input » et « PUT output ».
- Le transfert asynchrone. Une alarme est envoyée de façon asynchrone à une application en cas de survenue d'un problème quelconque. Un exemple d'alarme est la détection d'un début d'incendie. On doit pouvoir collecter des alarmes ou « TRAP ».

Que nous offre SIP pour cela ? Tout ce qu'il nous faut. Suivant les RFC 3261, 3265 et 3428, on peut utiliser :

- Le message SIP MESSAGE pour des actions PUT ou GET. Ce message a été originellement introduit pour la messagerie instantanée (RFC 3428).
- Les messages SIP SUBSCRIBE et NOTIFY les événements de type TRAP. Ces messages sont originellement utilisés pour la notification de présence (RFC 3265).

### **SIP vs SNMP**

Le protocole SNMP (*Simple Network Management Protocol*) a été présenté dans le magazine Linux Magazine numéro 43 d'octobre 2002 ainsi que sa mise en oeuvre pour le contrôle par Internet de systèmes électroniques.

SNMP offre les mêmes possibilités d'échange que SIP dans un contexte domotique. Il possède :

- Les messages SNMP PUT et GET pour des actions PUT et GET.
- Le message SNMP TRAP les événements de type TRAP.

Mais, SNMP par rapport à SIP possède des défauts majeurs :

- SNMP est un protocole complexe malgré son nom.
- Il est difficile à utiliser. L'extension d'un agent SNMP est compliquée.
- SNMP n'est pas un protocole très sécurisé surtout dans sa version 1.
- Un agent SNMP n'a pas une empreinte mémoire faible, ce qui est un handicap pour un système embarqué avec peu de mémoire.
- Il n'y a pas d'applications clientes populaires. SIP en a de nombreuses (*gaim* par exemple).

Le protocole SIP s'impose de lui-même dans un contexte domotique et de convergence électronique-informatique-réseaux.

### **HomeSIP : plateforme matérielle**

La plateforme matérielle HomeSIP est présentée figure 4.

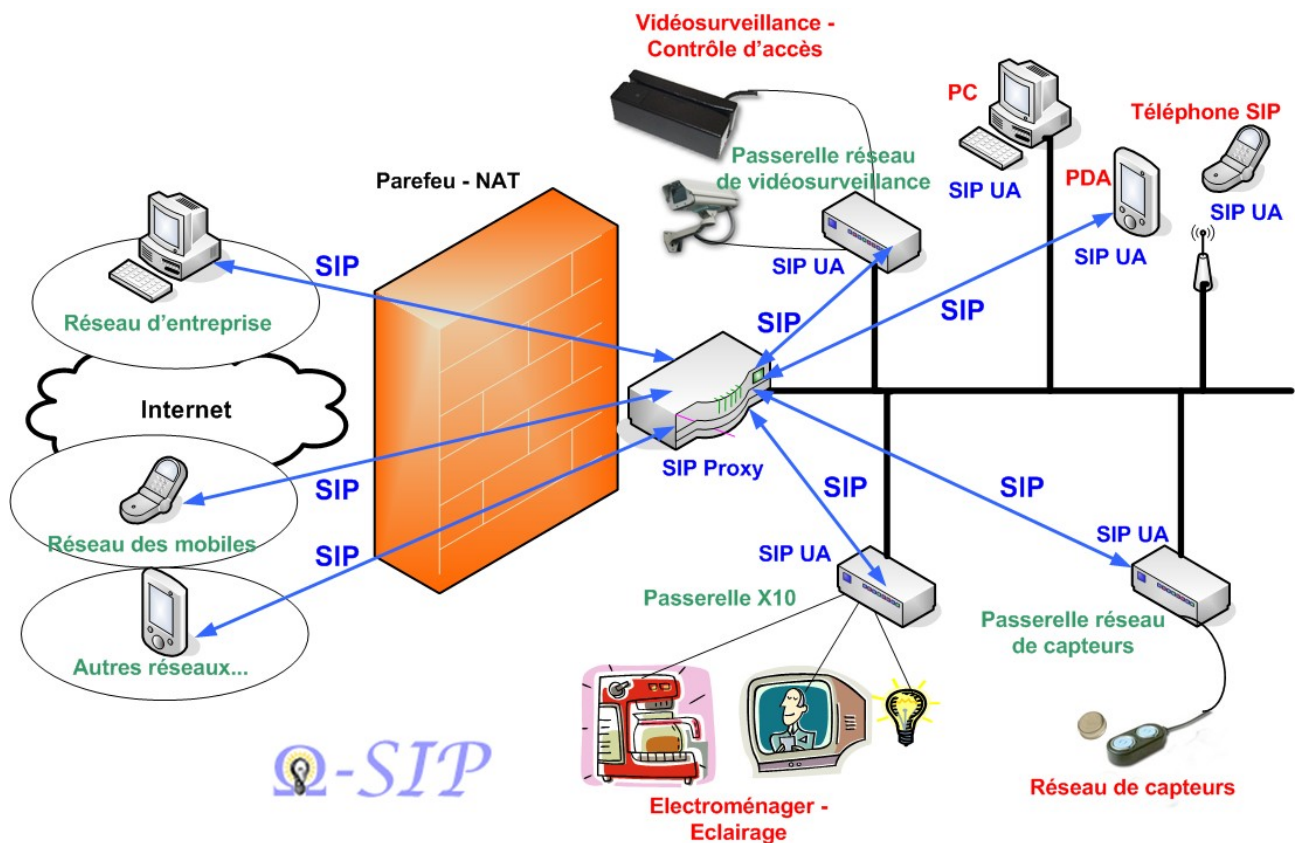


Figure 4 : Plateforme matérielle HomeSIP

Elle se greffe sur la plateforme SIP actuelle de l'ENSEIRB.

Les périphériques SIP utilisés sont des PC, des téléphones SIP ainsi que des PDA.

Seules, deux passerelles SIP/réseau de capteurs sont actuellement déployées et les développements logiciels sont en cours.

Il est prévu à terme d'intégrer les passerelles SIP/X10 pour la commande d'appareils électriques et les passerelles SIP/réseau de surveillance...

Le réseau Ethernet de l'ENSEIRB sert d'ossature à la plateforme HomeSIP.

Revenons à la passerelle SIP/réseau de capteurs. Cette passerelle est composée de matériels du commerce pour ne pas avoir des développements hardware à réaliser. Elle se compose :

- D'une carte ARM9 d'Eukréa. Cette carte est présentée en détail dans l'article d'Eric Benard. Son grand avantage est qu'elle supporte bien sûr Linux mais qu'elle possède de nombreuses E/S pour pouvoir connecter différents capteurs et actionneurs : liaisons série, bus I2C, E/S, bus USB... La carte possède aussi une interface réseau pour pouvoir remplir son rôle de passerelle.
- De capteurs de température *iButton*® DS1920 de Dallas Semiconductor avec une interface pour les connecter à une liaison série. L'interface de pilotage d'un *iButton*® étant standard, il est possible d'en rajouter d'autres...

## **HomeSIP : développements logiciels**

Les développements logiciels s'articulent actuellement autour de la passerelle SIP/réseau de

capteurs.

Il convient pour cela de :

- Porter Linux sur la carte cible ARM9. Cela a déjà été fait par Eukréa.
- Trouver une pile SIP à faible empreinte mémoire et la porter sur la carte cible. A défaut, il faut développer une pile SIP.
- Développer les applications Linux embarqué pour piloter les capteurs de température connectés à la carte cible.

### Portage d'une pile SIP pour la passerelle

Il existe différentes piles SIP libres que l'on peut utiliser sachant qu'il convient de privilégier celles qui sont développées en langage C ou C++.

Après différents tests, le choix s'est porté sur la pile GNU oSIP et son extension eXosip développées par Aymeric Moizard.

La pile oSIP étant écrite en langage C, elle est fortement portable et à faible empreinte mémoire. eXosip est une bibliothèque de fonctions (*Application Programming Interface*) basée sur oSIP pour faciliter l'écriture des applications SIP. L'API eXosip a bien sûr été privilégiée ici.

Le PC hôte est sous Linux Fedora Core 4. Son nom est `host` !

La cible est sous Linux embarqué. C'est la passerelle SIP/réseaux de capteurs. Son nom est `arm01` !

Après installation du compilateur croisé GNU C/C++ `gcc` pour processeur ARM fourni par Eukréa, le portage d'oSIP et d'eXosip est alors possible...

Environnement de travail :

```
$ cd
$ pwd
/home/guest
$ cd arm
$ ls
goarm_eXosip  iButton      libeXosip2-2.2.2.tar.gz  libosip2-2.2.2.tar.gz
goarm_osip    src
```

Décompression d'oSIP et d'eXosip :

```
$ tar -xvzf libosip2-2.2.2.tar.gz
$ ln -s libosip2-2.2.2 libosip
$ tar -xvzf libeXosip2-2.2.2.tar.gz
$ ln -s libeXosip2-2.2.2 libeXosip
$ ls
goarm_eXosip  libeXosip                libosip                    src
goarm_osip    libeXosip2-2.2.2         libosip2-2.2.2
iButton       libeXosip2-2.2.2.tar.gz  libosip2-2.2.2.tar.gz
```

Compilation croisée pour processeur ARM9 d'oSIP :

```
$ cat ./goarm_osip
cd ./libosip
CC=arm-linux-gcc CFLAGS=-O2 ./configure --prefix=$HOME/arm --disable-trace --
disable-debug --host=arm-linux
make
make install

$./goarm_osip
```

```
bla bla bla...
```

### Compilation croisée pour processeur ARM9 d'eXosip :

```
$ cat ./goarm_eXosip
cd ./libeXosip
CC=arm-linux-gcc CFLAGS=-O2 ./configure --disable-josua --prefix=$HOME/arm --
disable-trace --disable-debug --host=arm-linux
make
make install
$ ./goarm_eXosip
bla bla bla...
```

On a au final les bibliothèques oSIP et eXosip pour la cible et les fichiers *include* pour la compilation croisée de nos applications Linux embarqué :

```
$ ls
bin          iButton  libeXosip          libosip          man
goarm_eXosip include  libeXosip2-2.2.2  libosip2-2.2.2  src
goarm_osip  lib      libeXosip2-2.2.2.tar.gz  libosip2-2.2.2.tar.gz
```

Installation des bibliothèques oSIP et eXosip sur la cible arm01 en utilisant un montage NFS du répertoire de travail du PC host depuis la cible arm01 :

côté host :

```
$ /usr/sbin/exportfs
/home/guest      arm01
/home/guest      arm02
```

côté arm01 :

```
# mount -t nfs host:/home/guest /mnt
# cd /mnt
# ls
CPUAT91      HomeSIP      bin          iButton      src
Desktop      arm          doc          packages
# cp -r /mnt/arm/lib /
```

### Contrôle des capteurs de la passerelle

Pour les capteurs, on met en oeuvre ici les capteurs de température *iButton*® DS1920. Dallas Semiconductor fournit une bibliothèque libre de fonctions C pour piloter ces capteurs à partir d'un port série moyennant l'usage d'un adaptateur bus *I-Wire*®/liaison série (DS9097U-S09) :

Décompression de la bibliothèque pour le DS1920 :

```
$ cd iButton
$ unzip ulinuxgnu300.zip
```

L'application `temp.c` fournie est modifiée comme suit pour son utilisation sur la cible arm01 :

```
#include <stdlib.h>
#include <stdio.h>
#include "ownet.h"
#include "temp10.h"
#include "findtype.h"

#define MAXDEVICES          2

// global serial numbers
uchar FamilySN[MAXDEVICES][8];
int family_code;

int main(int argc, char **argv) {

    float current_temp;
```

```

int i = 0;
int NumDevices=0;
int portnum = 0;

// check for required port name
if (argc != 2) {
    printf("Error. Syntax: ./gettemp_ds1920 /dev/ttySx\n");
    exit(1);
}

// attempt to acquire the 1-Wire Net
if((portnum = owAcquireEx(argv[1])) < 0)
    exit(1);

// Find the device(s)
NumDevices = FindDevices(portnum, &FamilySN[0], 0x10, MAXDEVICES);

// read the temperature and print serial number and temperature
for (i = 0; i < NumDevices; i++) {

    if (ReadTemperature(portnum, FamilySN[i], &current_temp)) {
        PrintSerialNum(FamilySN[i]);
        printf("      %5.1f \n", current_temp);
    }
    else {
        printf("      Error reading temperature, verify device present:\n");
        // release the 1-Wire Net
        owRelease(portnum);
        exit(1);
    }
}

owRelease(portnum);
exit(0);
}

```

L'application temp.c est «cross-compilée» pour la cible arm01 pour donner l'exécutable

```

gettemp_ds1920 :
$ cat ./go_ds1920
make temp LINKFILE=linuxlnk.o CC=arm-linux-gcc
mv ./temp gettemp_ds1920
$ ./go_ds1920
bla bla bla...

```

L'application Linux embarquée gettemp\_ds1920 est copiée sur la cible arm01 puis testée :

```

# cp /mnt/arm/iButton/gettemp_ds1920 .
# ./gettemp_ds1920 /dev/ttyS2
6B000800BC7BAF10      25.1
# echo $?
0

```

### Exemple 1 : mise en oeuvre du message SIP MESSAGE

Le premier exemple de mise en oeuvre du protocole SIP est l'envoi périodique de la température mesurée par le capteur DS1920. C'est un exemple basique basé sur l'utilisation du message SIP MESSAGE.

Un agent SIP send\_im embarqué sur la cible arm01 envoie un message instantané vers un agent SIP du PC hôte. On utilisera pour cela l'agent SIP *gaim* côté PC mais aussi l'agent *josua* fourni avec oSIP. Cela correspond à la première méthode pour récupérer des informations en provenance de capteurs par SIP.

Le code source de l'application send\_im.c est le suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <syslog.h>
#include <pthread.h>
#include <eXosip2/eXosip.h>

main (int argc, char *argv[])
{
    char buf[256];
    osip_message_t *message;
    int i;
    eXosip_event_t *event;
    int port = 5060;

    // For trace
    // osip_trace_initialize(6, stdout);

    if(argc != 4) {
        printf("send_message from(sip:root@target) to(sip:root@host) info\n");
        exit(-1);
    }

    if (eXosip_init ()) {
        printf("eXosip_init failed\n");
        exit (1);
    }

    i = eXosip_listen_addr (IPPROTO_UDP, NULL, port, AF_INET, 0);
    if (i!=0) {
        eXosip_quit();
        printf("Could not initialize transport layer\n");
        exit (1);
    }

    eXosip_force_masquerade_contact ("localhost");

    i = eXosip_message_build_request (&message, "MESSAGE", argv[2], argv[1],
    NULL);
    if (i != 0) {
        printf("eXosip_message_build_request failed\n");
        exit (1);
    }

    osip_message_set_expires (message, "120");
    strncpy(buf, argv[3], sizeof(buf));
    osip_message_set_body (message, buf, strlen (buf));
    osip_message_set_content_type (message, "text/plain");

    eXosip_lock ();
    i = eXosip_message_send_request (message);
    if (i != 0) {
        printf("eXosip_message_send_request failed\n");
        exit (1);
    }
    eXosip_unlock ();

    // for tracing stack events
    while(1) {
        if (!(event = eXosip_event_wait (0, 0))) {
```

```

        usleep (10000);
        continue;
    }
    eXosip_automatic_action ();

switch (event->type) {
    case EXOSIP_MESSAGE_PROCEEDING:
        break;
    case EXOSIP_MESSAGE_ANSWERED:
        printf("EXOSIP_MESSAGE_ANSWERED\n");
        exit(0);
        break;
    case EXOSIP_MESSAGE_REDIRECTED:
        break;
    case EXOSIP_MESSAGE_REQUESTFAILURE:
        break;
    case EXOSIP_MESSAGE_SERVERFAILURE:
        break;
    case EXOSIP_MESSAGE_GLOBALFAILURE:
        break;
    default:
        printf("received eXosip event (type, did, cid) = (%d, %d,
%d)", event->type, event->did, event->cid);
        break;
}
eXosip_event_free (event);
}
}

```

L'application `send_im.c` est « cross-compilée » pour la cible `arm01` pour donner l'exécutable `send_im`:

```

$ arm-linux-gcc -O2 -g -I$HOME/arm/include -L$HOME/arm/lib -DENABLE_TRACE
send_im.c -o send_im -lexosip2 -losip2 -losipparser2 -lpthread

```

L'application Linux embarqué `send_im` est copiée sur la cible `arm01` :

```

# cp /mnt/arm/src/send_im .

```

Le lien entre l'application `gettemp_ds1920` et l'application `send_im` est fait grâce un simple *shell script* `gotemp_ds1920` qui est ensuite exécuté sur la cible `arm01` :

```

# cat gotemp_ds1920
#!/bin/sh
while [ 1 ]
do
    set `./gettemp_ds1920 /dev/ttyS2`

    if [ "$?" == "0" ]; then
        ./send_im sip:root@`hostname` sip:guest@host $2
    fi
done
# ./gotemp_ds1920

```

L'échange de messages SIP doit être celui de la figure 5.

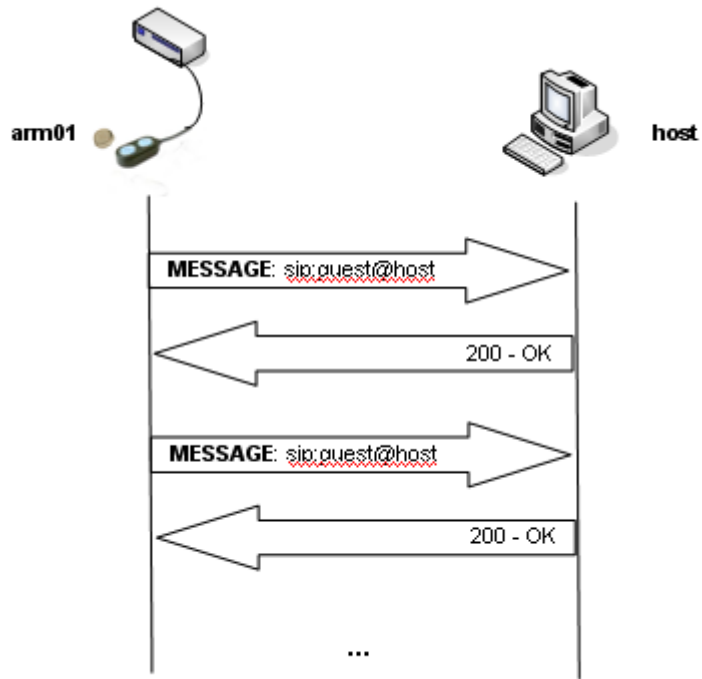


Figure 5 : Echanges de messages SIP généré par send\_im

L'outil *ethereal* nous fournit les traces des échanges de messages SIP de la figure 6 ; ce qui est bien conforme à ce qui est attendu.

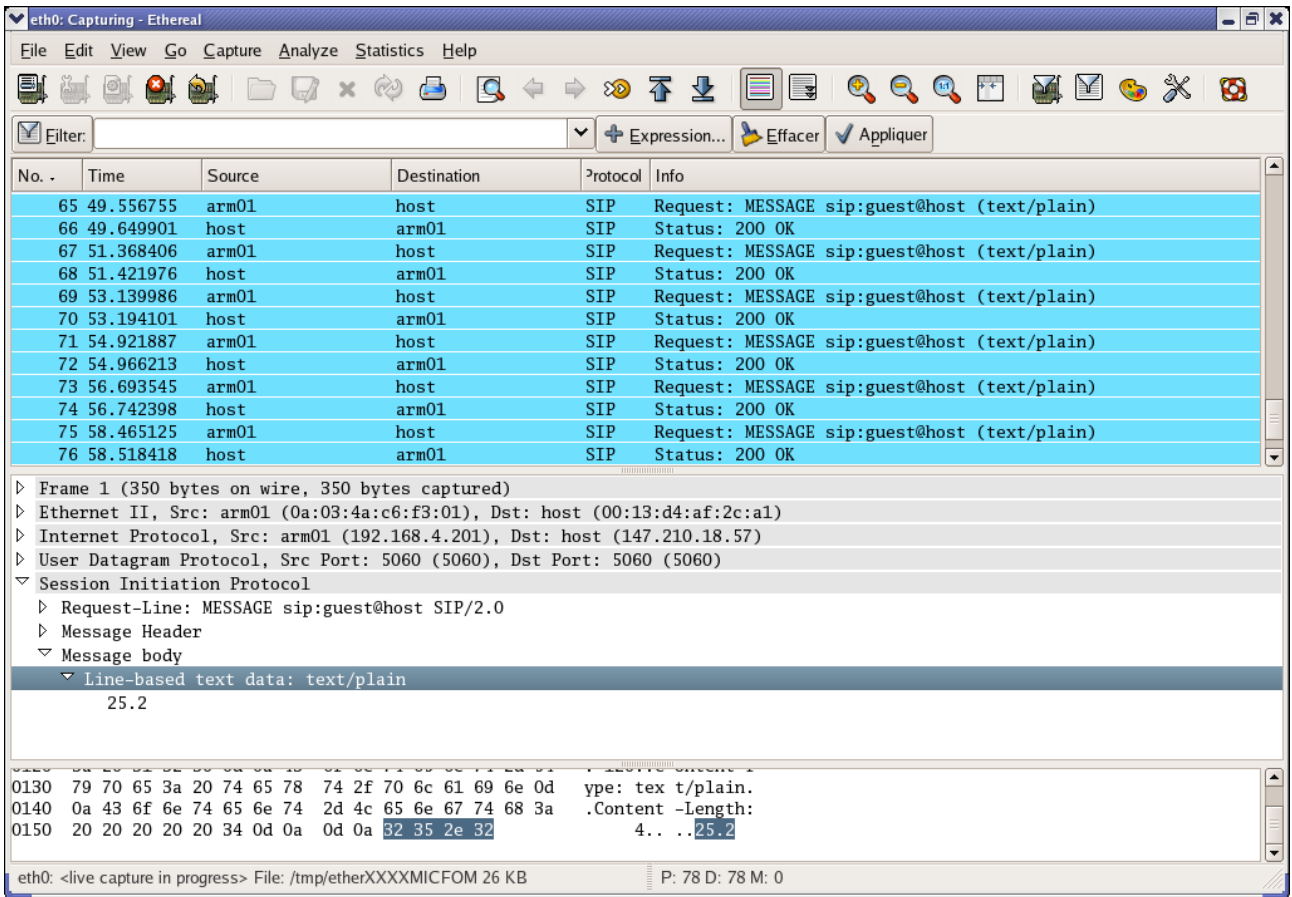


Figure 6 : Traces générées par l'application send\_im

La figure 7 montre les messages instantanés récupérés par *gaim*.

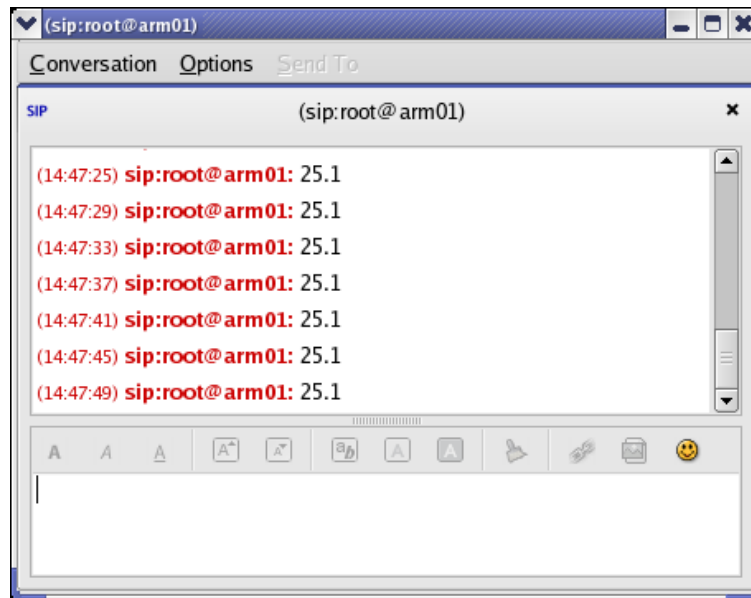


Figure 7 : Messages instantanés collectés par *gaim* (25,1 °C)

Enfin, la figure 8 montre les messages instantanés récupérés par *josua* en provenance des 2

passerelles arm01 et arm02.

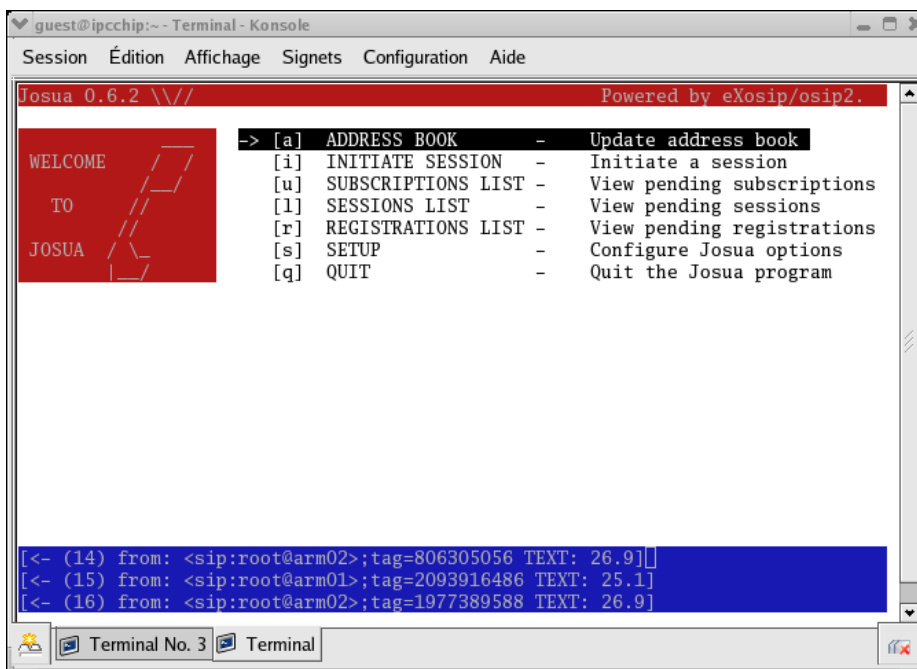


Figure 8 : Messages instantanés collectés par *josua* (25,1 °C et 26,9 °C)

### Exemple 2 : mise en oeuvre des messages SIP SUBSCRIBE/NOTIFY

Ce dernier exemple de mise en oeuvre du protocole SIP est l'envoi périodique de la température mesurée par le capteur après souscription. C'est un exemple basé sur l'utilisation du couple de messages SIP SUBSCRIBE/NOTIFY.

Un agent SIP s'inscrit (SUBSCRIBE) auprès de la passerelle pour recevoir pendant une durée déterminée la valeur courante du capteur de température. La passerelle lui renvoie durant toute la durée de souscription la valeur courante de la température (NOTIFY).

Une application `send_notify` embarquée sur la cible arm01 envoie l'information température à une application agent SIP `send_subscribe` du PC host. L'exemple étant plus compliqué que le précédent, on pourra récupérer les fichiers sources à l'adresse donnée dans la bibliographie.

L'échange de messages SIP doit être celui de la figure 9.

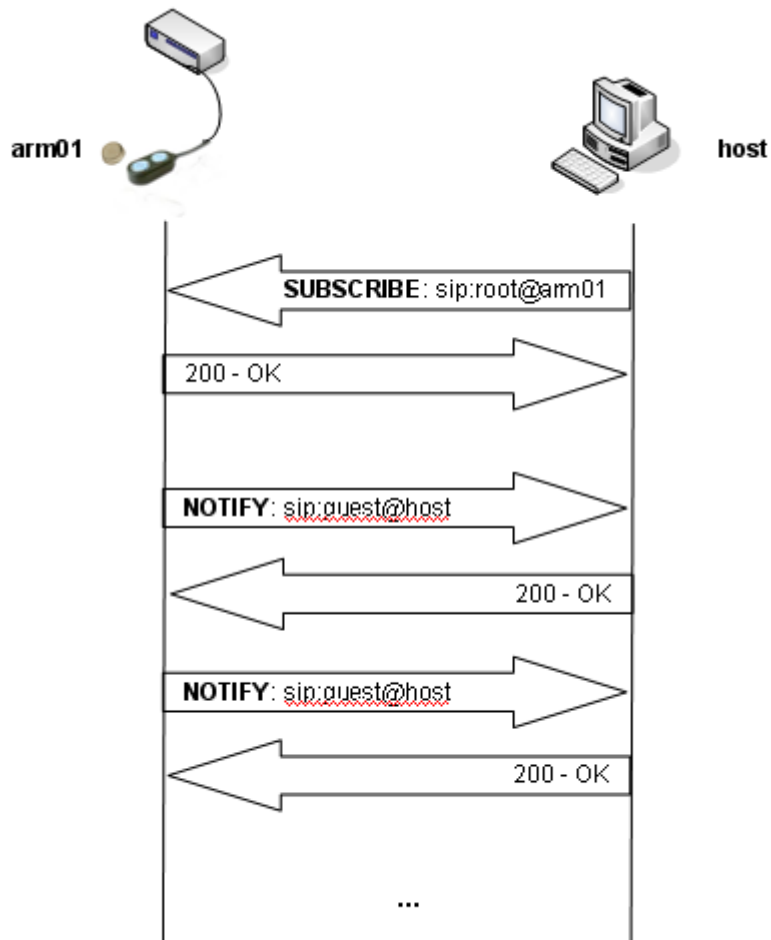


Figure 9 : Echange de messages SIP généré par `send_notify`

L'outil *ethereal* nous fournit les traces des échanges de messages SIP de la figure 10 ; ce qui est bien conforme à ce qui est attendu. La réponse 101 (*Dialog Establishment*) est informationnelle et facultative.

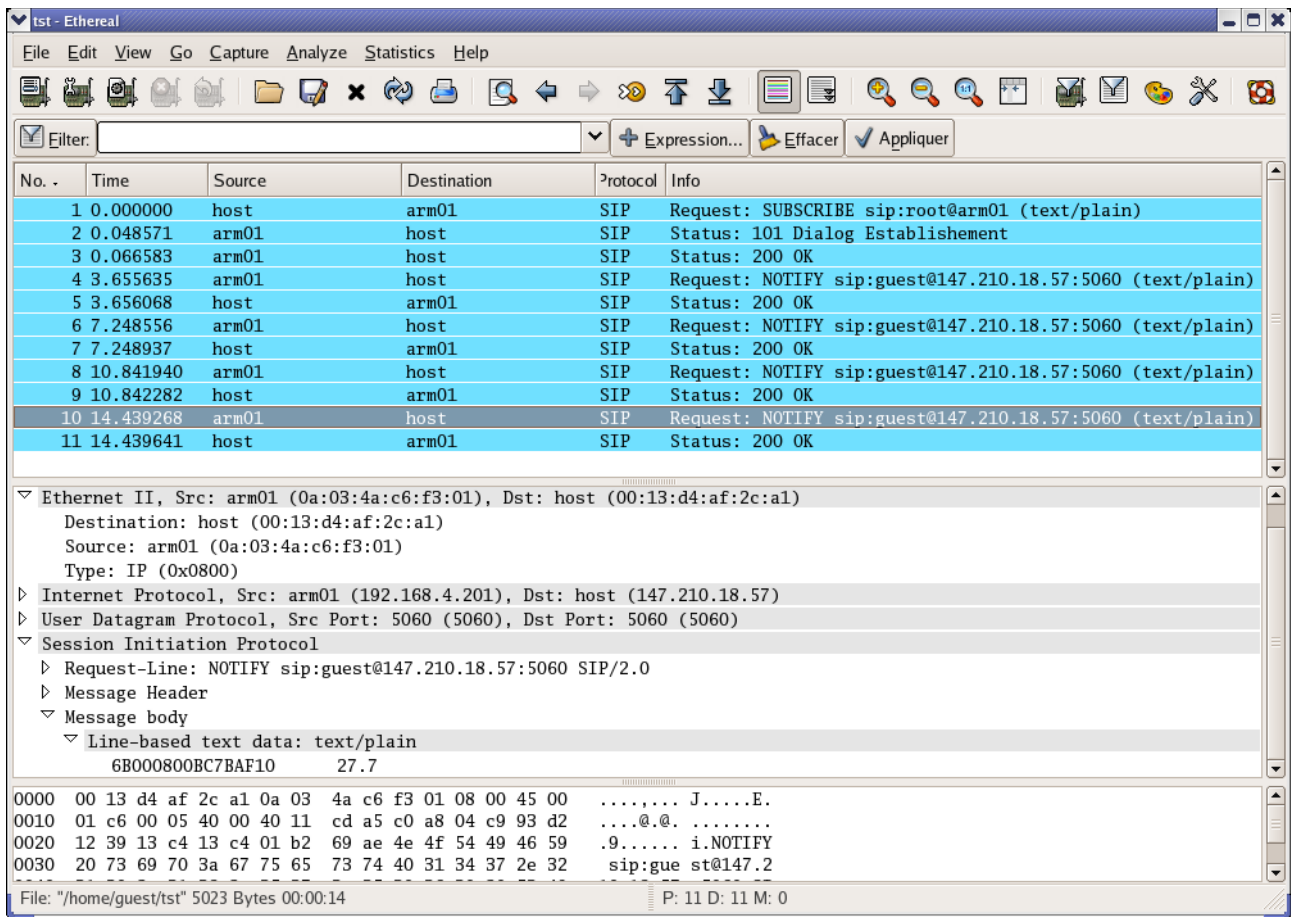


Figure 10 : Traces générées par l'application send\_notify (27,7 °C)

## Conclusion

A travers cet article, nous avons pu voir la mise en oeuvre du protocole SIP pour le pilotage de capteurs. La plateforme HomeSIP a été aussi décrite ainsi que les objectifs attendus. Des exemples pratiques ont été proposés et testés afin de démontrer la mise en application du concept.

Lancé début 2006, le projet HomeSIP en est à ses débuts et il reste beaucoup de choses à faire comme l'intégration des autres passerelles du point de vue matériel, le développement des applications embarquées, la structuration des échanges en utilisant XML par exemple et enfin le développement d'un langage de type DSL pour spécifier et créer des services SIP pour la domotique.

Pour terminer, il est toujours étonnant et « magique » de constater la facilité d'intégration de logiciels libres sans perte de temps dans un projet Linux ou Linux embarqué !

## Bibliographie

- Le projet HomeSIP : <http://www.enseirb.fr/cosynux/HomeSIP/>
- Le portail sur les ressources SIP : <http://www.cs.columbia.edu/sip/>
- SNMP. Administrez facilement votre réseau. P. Kadionik. Linux Magazine numéro 43. Octobre 2002

- Carte ARM9 d'Eukréa : <http://www.eukrea.com/>
- Page sur les *iButton*<sup>®</sup> de Dallas Semiconductor : <http://www.maxim-ic.com/products/ibutton/products/ibuttons.cfm>
- Ressources logicielles pour *iButton*<sup>®</sup> : <http://www.maxim-ic.com/products/ibutton/software/1wire/wirekit.cfm>
- Bibliothèque en langage C pour *iButton*<sup>®</sup> : [http://files.dalsemi.com/auto\\_id/public/ulinuxgnu300.zip](http://files.dalsemi.com/auto_id/public/ulinuxgnu300.zip)
- Document de comparaison des différentes piles SIP libres : <http://www.huisetalage.nl/sip/stacks.pdf>
- Pile GNU SIP oSIP : <http://www.gnu.org/software/osip/>
- Documentation oSIP : <http://www.antisip.com/documentation/osip2/>
- Bibliothèque eXosip : <http://savannah.nongnu.org/projects/exosip/>
- Documentation eXoSIP : <http://www.antisip.com/documentation/eXosip2/>
- Fichiers sources de l'article disponibles à : <http://www.enseirb.fr/~kadionik/embedded/hs25/homesip.src.tar.gz>