Comprendre et programmer le protocole SMTP ?

L'application la plus utilisée sur l'Internet est le mèl. Pour acheminer les informations du rédacteur au destinataire, les serveurs utilisent le protocole SMTP pour dialoguer.

Le mois dernier, nous avons vu un premier protocole texte utilisé sur l'Internet : HTTP. Les caractéristiques principales de ce protocole sont : connexion TCP, mode client/serveur et message texte. Eh bien pour le protocole SMTP, c'est exactement la même chose. Simple Mail Transfert Protocol : protocole simple de transfert de courrier. Ce protocole est tellement simple qu'avec le développement d'Internet et du courrier électronique, il a été nécessaire de créer un nouveau standard ESMTP, E pour Extended.
Il est devenu difficile de trouver un serveur opérationnel n'utilisant pas la version étendue de ce protocole et tous les exemples qui suivent seront donc donnés avec un serveur supportant cette évolution. Cet article se propose de vous initier aux bases du protocole SMTP, et à la programmation de clients simples en langage C et java.

Envoyer un email

D'abord, en français on ne dit pas "email", ni "mail", on dit "mèl", on peut dire également "courriel". Mais, pour la bonne compréhension du texte, je continuerai à écrire "email" ou "mail".
Le processus de transfert d'un email met en œuvre une chaîne d'application et de protocole plus large que le simple protocole SMTP. Commençons par quelques définitions :

MUA : Mail User Agent, c'est le client de messagerie (KMail, Evolution, etc.).

MTA : Mail Transfert Agent, c'est l'agent qui va transférer votre mail vers le serveur chargé de la gestion des emails de votre destinataire. Dans la pratique, le courrier peut transiter par plusieurs MTA.

MDA : Mail Delivery Agent est le service de remise du courrier dans les boîtes aux lettres des destinataires.

Donc si on résume, le MUA transfert l'email à un MTA qui le transfère au MTA du destinataire (ou à un MTA intermédiaire) qui le passe au MDA chargé de stocker l'email dans la boite aux lettres du destinataire. Dans la pratique le MUA établit une connexion SMTP avec un MTA qui contacte via SMTP le MTA du destinataire qui est aussi un MDA.

Ensuite libre au destinataire d'utiliser une commande simple (mail) ou un outil plus complexe (KMail, …) pour lire ces emails, utilisant le protocole POP ou IMAP (que nous aurons la joie de découvrir dans quelques semaines).

Un protocole client serveur

A l'instar de http, SMTP est un protocole client serveur en mode texte Il est défini dans la RFC 821. Cette dernière écrite en 1982 par Jon Postel (la légende, le père de l'Internet), se base sur une infrastructure bien moins performante que celle dont on dispose aujourd'hui. Il est donc prévu que les caractères transmis soient codés sur 7bits, oubliez les caractères accentués et autre gothique. Pour se connecter à un serveur SMTP (celui mis à disposition par votre fournisseur d'accès Internet fera parfaitement l'affaire), un simple telnet est utilisé :

telnet smtp.wanadoo.fr 25

Le port 25 est celui généralement utilisé par le serveur pour recevoir une connexion. Vous retrouverez cette valeur si vous jetez un œil dans le fichier /etc/services. Une fois la connexion établie, le serveur s'identifie.

220 mel-rta10.wanadoo.fr ESMTP Service (6.5.007) ready

On notera au passage que c'est un serveur ESMTP. Ensuite il convient d'être poli et de s'identifier en déclinant son nom de domaine:

HELO salemioche.com

Cette première opération sert à vérifier que la connexion est établie correctement et que les deux participants peuvent s'échanger des informations. (le « HELLO protocole » est une technique très répandue pour valider la qualité d'un lien). Quant une opération se déroule bien, le serveur répond avec le code 250.

250 mel-rta8.wanadoo.fr

Nous allons ensuite envoyer un message à nicolas@salemioche.com en tant que linuxmag@salemioche.com.

MAIL FROM: <linuxmag@salemioche.com>
250 MAIL FROM:<linuxmag@salemioche.com> OK
RCPT TO: <nicolas@salemioche.com>
250 RCPT TO:<nicolas@salemioche.com> OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
bonjour,
ceci est un message de linuxmag pour nicolas.
.
250 <3D8011E600743103> Mail accepted
QUIT
221 mel-rta7.wanadoo.fr QUIT

Et je peux vous confirmer que j'ai bien reçu ce mail quelques secondes plus tard.

Spamming

Toutes les informations proposées ici sont destinées à la compréhension du protocole et à la réalisation de petits outils à usage personnel. La maîtrise de cette technique ne doit pas être utilisée pour le spam, courriers non sollicités envoyés à de très nombreuses personnes et/ou de manière répétitive. Ces activités sont contraire à la netiquette et peuvent faire l'objet d'actions répressives par les fournisseurs d'accès internet. Association des fournisseurs d'accès et de service internet : http://www.afa-france.com.

MAIL FROM:

Rentrons un peu dans le détails, une fois le HELO de courtoisie échangé (il n'est pas toujours nécessaire), on utilise la commande 'MAIL FROM:' suivi du chemin de retour. Il s'agit en fait de l'endroit ou sera envoyé le message s'il n'est pas possible de le délivrer au destinataire. Dans la pratique, on utilise ici sa propre adresse email entourée de chevrons <>. Par exemple :

MAIL FROM: <nicolas@salemioche.com>

Après chaque commande on utilise les caractères 'carriage return' et 'line feed' (respectivement '\r' et '\n' dans les langages de programmation usuels). Cette commande doit être la première pour débuter une session d'envoi d'un message.

RCPT TO:

La commande 'RCPT TO:' est suivi du chemin d'accès du destinataire. On parle ici de chemin et nom d'adresse car il est possible de spécifier plusieurs hôtes avant l'adresse. Un chemin a la forme "@UN,@DEUX:JOE@TROIS", dans laquelle UN, DEUX, et TROIS sont des noms d'hôtes. Cette forme est employée dans le but d'accentuer la différence formelle entre une adresse et une route. La boîte aux lettres est une adresse absolue, la route est une information permettant d'y accéder. Ces deux concepts doivent toujours être dissociés, cependant dans la vraie vie de l'utilisateur lambda à la maison cela n'arrive jamais (pour plus détails consultez la rfc). Je vais donc me contenter d'utiliser une simple adresse. On peut utiliser la commande RCPT plusieurs fois pour envoyer un même message à différentes adresses. Exemple :

RCPT TO: <nicolas@salemioche.com>
RCPT TO: <nicolas@salemioche.net>

Ce message sera donc délivré aux 2 adresses. Le serveur valide l'adresse en renvoyant le code 250 après chaque ligne de commande.

DATA

La commande nécessite deux étapes. On passe la commande au serveur, il répond :

354 Start mail input; end with <CRLF>.<CRLF>

La saisie du message peut commencer. Pour l'envoyer, saisissez uniquement et en début de ligne le caractère '.' (comme l'indique la réponse du serveur). Si vous souhaitez qu'une ligne ne contienne qu'un seul point il faut en mettre deux. Ainsi le message :

Bonjour
..
.

sera reçu par le destinataire :

Bonjour
.

Sur la réception de la ligne avec le '.', le serveur valide la requête et envoie l'email. Si tout se passe bien il renvoie le code 250. Attention, si l'adresse email est invalide ou si la boite aux lettres du destinataire est inexistante ou encore pleine, le message ne sera pas délivré et vous recevrez quand même le code 250. Comme nous le verrons plus loin, la section DATA peut être composée d'un entête et d'un corps séparé par une ligne blanche, pour éviter les mauvaises surprises, si vous ne mettez pas d'entête à vos messages commencez les par une ligne blanche sous peine de perdre le première ligne d'information.

QUIT

Et sans surprise la commande 'QUIT' permet de terminer la transaction.

COMMANDES

Il existe d'autres commandes, mais elles ne sont pas toujours disponibles sur les serveurs. On notera pêle-mêle HELP, liste les commandes disponibles sur le serveur, VRFY, vérifie une adresse destinataire, RSET, pour annuler les commandes passées et EXPN pour vérifier la disponibilité d'une liste de distribution et récupérer la liste des adresses associées.

Le message

Si vous avez fait les tests précédents vous avez remarqué que lorsque vous lisez le mail, les informations 'from:', 'to:', 'subject:' (de, à et sujet en français) ne sont pas renseignés correctement. En effet, il faut distinguer les informations fournit lors du passage des commandes MAIL et RCPT qui permettent d'acheminer l'information, du contenu du message. C'est un peu comme si le facteur vous livrait une lettre en enlevant l'enveloppe au moment de la glisser dans votre boite aux lettres, à moins que l'expéditeur ait inscrit son adresse et la votre sur la lettre vous n'avez aucun moyen de savoir qui l'a envoyée. On va donc préciser dans le message (la section DATA), un certain nombre de champs. Ces champs doivent figurer au tout début du message et

forment l'entête. Si vous saisissez une ligne blanche pour commencer votre message, ils ne seront pas pris en compte. Voici un petit exemple de session utilisant ces champs :

220 mel-rta7.wanadoo.fr ESMTP Service (6.5.007) ready
HELO salemioche.com
250 mel-rta7.wanadoo.fr
MAIL FROM:<nicolas@salemioche.com>
250 MAIL FROM:<nicolas@salemioche.com> OK
RCPT TO:<nicolas@salemioche.net>
250 RCPT TO:<nicolas@salemioche.net> OK
DATA
354 Start mail input; end with <CRLF>.<CRLF>
from: "Nicolas JEAN" <nicolas@salemioche.com>
to: "Nicolas at salemioche dot net" <nicolas@salemioche.net>
subject: email avec champs dans l'entete

coucou, ceci
est
un message
.
250 <3D8011E6007C45C8> Mail accepted
QUIT
221 mel-rta7.wanadoo.fr QUIT

Les champs sont formés d'une façon classique, le nom du champ immédiatement suivi de ':', un espace et la valeur du champ. Des champs pourront être ajoutés par les différents MTA pour ajouter des informations sur le routage du message jusqu'à son destinataire. Ces champs servent donc au MUA pour afficher l'email, le renseigner et permettre au destinataire d'y répondre. L'entête du message est séparé par une ligne blanche du corps du message. La RFC 822 décrit la liste de ces champs, et voici les plus courant :

From : from est suivi de l'adresse et de la route de l'expéditeur, on note ici la différence de notion entre adresse et route (entre chevrons). Théoriquement on peut donc avoir :

From: "nicolas@salemioche.com" <@smtp.wanadoo.fr,@smtp.salemioche.com:nicolas@salemioche.com"

Cet usage standard à finalement été détourné pour devenir :
From: "Nickname" <adresse>

To : liste des destinataires principaux, on retrouve le format "Nick" <adresse>. Les adresses se suivent et sont séparées par une virgule. Exemple :

To: "com" <nicolas@salemioche.com>, "net" <nicolas@salemioche.net>

Cc : liste des destinataires en "copy carbon". Le format est le même que pour le champ 'To'. On remarque que le protocole SMTP ne fait pas de différences entre destinataire principal (To) et destinataire en copie (Cc), chacun nécessite la même commande RCPT.

Bcc : liste des destinataires en copie cachée. Personnellement, je ne trouve que peu d'intérêt à ce champ. Les adresses contenues dans ce champ ne sont pas incluses dans les copies des messages envoyées aux destinataires principaux et secondaires (Cc). Des systèmes peuvent choisir d'inclure le texte de cette zone "Bcc" seulement dans la copie de l'auteur, alors que d'autres peuvent aussi l'inclure dans le texte envoyé à tous ce qui sont indiqué dans la liste "Bcc". Dans la plupart des cas je vous recommande de ne pas l'utiliser. Pour envoyer une copie cachée, on utilise la commande RCPT pour une adresse qui ne figurera pas dans le corps du message.

Subject : pour le sujet du message.

Return-Path : donne une adresse pour la réponse qui soit différente de l'adresse utilisée pour l'envoi (celle qui figure dans le champ 'To'). Quand le destinataire utilisera la fonction 'Répondre' de son outil de messagerie, c'est cette adresse qui sera utilisée.

Date : date du message, parfait pour antidater un message :-). Exemple de date : *Wed, 25 Sep 2002 17:39:28 +0200 (CEST).* Si vous ne spécifiez pas de valeur, le premier MTA s'en chargera généralement.

Le but de cet article étant d'écrire un outil d'envoi de mail, ces quelques champs sont largement suffisant, je vous renvoie à la RFC pour 'tuner' vos messages plus précisément.

De théorie assez à C

Passons à la pratique, un petit client en langage C pour envoyer par email le contenu d'un fichier. La gestion des erreurs a volontairement été supprimée afin de simplifier au maximum le code :

```
#include <stdio.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PACKET_SIZE  1024

void appli(char *f);
int writen(int fd, char *ptr, int n);
int readn(int fd, char *ptr, int n);

int  to_server_socket = -1;
char server_name[] = "smtp.wanadoo.fr";  /* nom du serveur SMTP*/
int  port = 25;
char helo[] = "HELO mon_domaine\n";
```

```
char from[] = "MAIL FROM: <monNom@monDOMAINE>\n";
char to[] = "RCPT TO: <destinataire@sonDomaine>\n";

char subject[] = "Subject: transfert ";

int main (int argc, char *argv[])
{
   struct sockaddr_in serverSockAddr;    /* addresse de la socket */
   struct hostent *serverHostEnt;        /* description du host serveur */
   unsigned long hostAddr;               /* addresse du serveur */
   char *filename;

   if ( argc == 2 ) filename = argv[1];
   else filename = NULL; /* use stdin */

   bzero((void *)&serverSockAddr,sizeof(serverSockAddr));     /* initialise a zero
serverSockAddr */
   /* converti l'adresse ip 9.100.1.1 en entier long */
   hostAddr = inet_addr(server_name);

   if ( (long)hostAddr != (long)-1)
     bcopy((void *)&hostAddr,(void *)&serverSockAddr.sin_addr,sizeof(hostAddr));
   else                                  /* si on a donne un nom  */
   {
     serverHostEnt = gethostbyname(server_name);
     bcopy((void *)serverHostEnt->h_addr,(void *)&serverSockAddr.sin_addr,serverHostEnt-
>h_length);
   }

   serverSockAddr.sin_port = htons(port);         /* host to network port  */
   serverSockAddr.sin_family = AF_INET;           /* AF_*** : INET=internet */
   /* creation de la socket */
   to_server_socket = socket(AF_INET,SOCK_STREAM,0);
   /* requete de connexion */
   connect(to_server_socket,(struct sockaddr *)&serverSockAddr, sizeof(serverSockAddr));
   appli(filename);

   /* fermeture de la connection */
   shutdown(to_server_socket,2);
   close(to_server_socket);
   return 0;
}

void appli (char *filename) {
char buf[PACKET_SIZE+1], *ptr;
FILE *bulk;
int nb;
```

```
        if (filename == NULL)  bulk = stdin;
        else  bulk = fopen(filename,"rb") ;
        buf[0] = 0x00;
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);
        writen(to_server_socket,helo,strlen(helo));
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);
        writen(to_server_socket,from,strlen(from));
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);
        writen(to_server_socket,to,strlen(to));
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);
        writen(to_server_socket,"data\n",6);
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);

        sprintf(buf,"%s %s\r\n\r\n", subject,
                (( filename == NULL ) ? "STDIN" : filename ) );
        writen(to_server_socket,buf,strlen(buf));
        /* le file */
        while ( !feof(bulk) ) {
          bzero((void *)buf, PACKET_SIZE+1);
          nb = fread(buf,sizeof(char), PACKET_SIZE, bulk);
          buf[nb] = 0x00;
          while ( (ptr = strstr(buf,"\r\n.\r\n") ) != NULL ) *(ptr+3) = '.';
          writen(to_server_socket,buf,nb);
        }

        writen(to_server_socket,"\r\n.\r\n",5);
        readn(to_server_socket,buf,PACKET_SIZE);
        printf(buf);
        fclose(bulk);
}

int writen(int fd, char *ptr, int n)
{
int nl, nw;
 nl = n;
 while ( nl > 0 ) {
  nw = write(fd, ptr, nl);
  if ( nw <= 0 ) return nw;     /*erreur*/
  nl -= nw;
  ptr += nw;
  }
```

```
 return (n-nl);
}


int readn(int fd, char *ptr, int n){
int nl, nr;
 nl = n;
 while ( nl > 0 ) {
  nr = read(fd,ptr,nl);
  if (nr < 0 ) return nr;     /*erreur*/
  else if ( nr == 0 ) break;
  nl -= nr;
  ptr += nr;
  if ( *(ptr-2) == '\r' &&  *(ptr-1) == '\n' )
    break;
 }
 *ptr = 0x00;
 return (n-nl);
}
```

On définit trois variables qui sont "server_name", le nom de votre serveur SMTP, "from", la commande MAIL et "to" pour la commande RCPT. Modifier ces adresses pour tenir compte de vos choix. Il suffit ensuite de compiler ce programme tout simplement :

```
 gcc –o client client.c
```

et de le lancer avec le nom du fichier à envoyer :

```
client client.c
```

La première partie du programme est dédiée à la connexion avec le serveur, ensuite le dialogue s'installe et l'on retrouve la succession de commandes/réponses du protocole. Le seul champ utilisé est 'subject', je vous laisse le loisir de compléter le message pour obtenir un email agréablement lisible dans votre outil de messagerie. Attention si votre fichier contient une ligne avec seulement un '.', la suite du fichier ne sera pas envoyée.

Et en java

Voici une classe qui permet d'envoyer un message :

```
import java.io.*;
import java.net.*;

class smtpClient {
  private String server, message, dest, origin;
```

```
private int port;
private Socket socket;

public smtpClient (String serv, int por, String org)
{
  server = serv;
  port = por;
  origin = org;
}

public void createMsg(String dst, String msg)
{
  message = msg;
  dest = dst;
}

public void sendMsg()
{
  PrintWriter to;
  BufferedReader from;
  String str;

  try {
    socket = new Socket(InetAddress.getByName(server),port);

    to = new PrintWriter(
              new BufferedWriter (
                 new OutputStreamWriter (
                    socket.getOutputStream())),true);
    from = new BufferedReader(
             new InputStreamReader (
                    socket.getInputStream()));

    str = from.readLine();
    to.println("helo plus.bas");
    str = from.readLine(); System.err.println(str);
    to.println("mail from: <"+origin+">");
    str = from.readLine(); System.err.println(str);
    to.println("rcpt to: <"+dest+">");
    str = from.readLine(); System.err.println(str);
    to.println("data");
    str = from.readLine(); System.err.println(str);
    to.println("subject: petit mot\r\n\r\n" + message + "\r\n.\r\n");
    str = from.readLine(); System.err.println(str);

    socket.close();
```

```
    } catch ( Exception e ) {}

  }

  public static void main ( String args[] )
  {
    smtpClient c = new smtpClient("SMTP SERVER",25,"org@salemioche.com");
    c.createMsg("dest@salemioche.com","Coucou");
    c.sendMsg();

  }
}
```

Vous devez configurer le serveur SMTP à utiliser, ainsi que les emails de l'auteur (org) et du destinataire (dest). Une fois la classe encodée de la façon suivante :

javac Client.java

Il ne reste plus qu'à faire notre envoi :

java Client

Vous allez recevoir un petit email 'coucou'. La classe est instanciée en donnant le nom du serveur à utiliser, le port utilisé et votre adresse email. Ensuite on peut passer le destinataire et le message à cet objet. Il ne reste plus qu'à l'envoyer. Cette méthode reprend la succession de commandes/réponses du programme en C. Attention aucune erreur n'est traitée. De même aucun champ de l'entête du message n'est renseignés.

Conclusion

Si tout est clair jusque là, vous avez une bonne vue d'ensemble sur le protocole SMTP et plus rien ne vous empêche à présent d'automatiser les envois de mails pour recevoir automatiquement des informations dans votre boite à lettres. L'envoi d'emails met en œuvre d'autres techniques tel que l'usage du DNS, l'encodage MIME, l'envoi de pièces jointes, le relaying, ESMTP… Je vous propose de nous retrouver dans trois mois pour une session SMTP avancée. En attendant, nous verrons comment récupérer ces jolis emails avec les protocoles POP3 et IMAP4, et l'encodage MIME.

Nicolas JEAN
http://www.nicolasjean.com