SMTP – Simple Mail Transfer Protocol

What is SMTP?

SMTP stands for *Simple Mail Transfer Protocol*. This protocol allows transmitting electronic mail over the Internet or any other network. The protocol itself is designed on a character basis. Thus the entire transfer of a mail can be examinated by a human beeing without having to decode first the data. This is the reason why mails can be send without major problem using a simple Telnet application. The entire process of sending a mail is describe here using a Telnet application as example base.

The communcation is always carried out over the port 25. A mailserver needs an open port 25 which is one of the major problems if ISPs are blocking those. In contrast to the HTTP service the SMTP service can not simply use another port to communice out of the box. This is due to the restricting nature of this protocol.

What is a MTA and what's his function?

The MTA is the middle man between the client and the server in a mail transfer scenario. It stands for *Mail Transfer Agent*. This piece of software is running on a mail server and has the responsibility to transfer mails from the client to the server. You can imagine an MTA like a postman delievering mails to your mailbox. But in this case the mailbox is not in front of your house. Beyond this MTAs have also a responsibility of checking mails depending on the restrictions and demands of the mail server they are running on.

The Process of Mail Transfer

Sending a mail to another person is not a difficult process but it's far from beeing a short process. The entire transfer process starts at the user who wishes to send a mail to his collegue. After writing the text he sends it off. At this point the mail client, which is usually something like *Mozilla Messanger*, *Outlook* or even more simple a Telnet application, takes over the mail and prepares it for sending. For this the mail client generates a mail header containing the informations needed to carry out the transfer process. This header contains informations like the adress of the person who send the mail, the adress of the person who shall receive the mail and additional informations like the subject of the mail or feedback options. Headers are dealt with in more detail later in this document. Any attachments are converted into text form too. It's a bit unbelievable at first that all binary files are not transmitted as binary files in a mail but it's true. This is also the reason why send binary files tend to be bigger than the original file itself. After packing together all informations the mail client is ready to initiate the process of delivering the mail to the mail server.

At first the mail client starts a communcation with the sending mail server on the port 25. A mail server will only accept mails on this port unless the mail server has been configurated to not do this. But it's highly unlikely you will ever see a mail server not running on port 25. This is the mail server where our user has an account and a mail box. This server is also called the *Relaying Server*. He has the responsibility to forward the mail to the correct mail server owning the mailbox we are looking for. If you send a mail you notice that you have to suply login informations before you can go on. This is due to the relaying process. Mail Providers do this to decrease the likelyhood of spammers using their mail server to distribute spam in large numbers. Thus only registred users can use the server to relay mails through it. There exist different methods of verifying a user trying to send a mail. The simplest method forces the mail client to check for new mails before sending. This way the server knows from which IP the user is trying to communicate with the server. The IP is marked as valid for a certain period of time (something around 15 to 30 minutes). During this time the mail client can then send mails through this server without further verification. Another possibility is using an authentification process. In this the mail client has to send authentification informations like username and password just after initiating the transfer process to verify it`s identity. These are the major versions of verifying the user. If you are using a Telnet application you do not need this step if you are the mail server yourself. Sounds a bit strange but you can play mail server yourself. The detailed transfer process is described later using this Telnet shema.

As soon as the mail server has verified that you are allowed to send mails using his services your mail client will transfer the entire mail to the mail server. The relaying mail server checks the receiver adress and hooks up with a DNS service to determine the IP of the mail server owning the user mentioned in the mail adress.

The mail adress looks always the same. The first part contains the name of the user on the mail server. Then, seperated by an @, the name of the mail server is specified. Thus rptd@gmx.net identifies the user *rptd* on the mail server named *gmx.net*. You recognizes this one perhaps, it's the GMX Mail Provider. Thus an email can tell you immediately where a certain user has it's account. A mail adress is only considered valid if all those parts are present.

Now the relying mail server has found out the IP of the server it has to send the mail to. Our relaying mail server now starts a communication on the port 25 mentioned above. The receiving mail server checks the mail send request of the relaying mail server for validity. At this point spam filters can already reject a mail based on the server trying to connect or the domain the server is residing in. Now that the receiving mail server is happy with the mail send request it checks the user part of the receiver mail adress to locate the mail box on the server. If it can not find that user the system will reject the mail and deliver an error report back to the sending mail server. Otherwise the receiving mail server accepts the mail transfer and stores the received mail unchanged in to the mail box just found. Finished with this the receiving mail server reports a success to the sending mail server and shuts down the communcation.

All this mail checkings and the final storing of the mail in the mail box is performed by the MTA described above. If a mail can not be delievered to a mail box because such a mail box does not exist or the mail box is not accessible or another error occurred then the MTA can decide the accept the mail nevertheless. In this case the MTA stores the mail in a temporary place and tries delievering the mail at a later time in the hope of beeing able to succed at this point. If the MTA is not able to deliver the mail after a certain time or number of tries it will generated a failure mail and send it back to the sending mail adress. These are those MAILER-DEAMON mails you have received once upon the time certainly. Thus the process of sending a mail is not reliable in the common way. There are services allowing secure mail transfer which are described later on.

The sending mail server will now report the success or failure back to the mail client. Sometimes this mail server can also keep a copy of the mail in the hope of beeing able to deliver it at a later time. After sending back the success report to the mail client the sending mail server shuts down the communcation and goes home.

After receiving a success or failure notice from the sending mail server the mail client will finish the mail transfer and another mail has found it's way to it's goal or sometimes not. By now the mail is only at the mail server of the receiver. To be able to read it the appropriate user has to fetch the mail.

Now we are the user who wants to fetch his new mail. For this he uses again his mail client and his relaying mail server. But this time he uses the so called POP3 protocol. This is short for *Post Office Protocl* and is the version 3. This protocol is similar to the SMTP protocol in that way it uses a pure character oriented communcation itself and that the communcation shema is comparable to the SMTP way of doing transfers. Another difference is that the POP3 protocol communcation between the mail client and the mail server is held on port 110. This is the common port for POP3 communcation but can be changed too if needed. The mail client connects thus to the mail server and aska for new mails. Again the mail server will check for the correct user to access the mailbox. If all is ok the mail server will send the just newly received mail to the users mail client. Shutting down the communcation our user is now able to read his new mail.

The Mail Tranfer Process in detail, explained at Telnet

Now we go over to the concrete transfer of a mail using a Telnet application as example. A Telnet application is in fact no more than a kind of simple one way chat between a client and a server. The client connects to the server exhibiting a couple of commands and the server answers to those commands with certain informations or actions. A very simple process thus the name '*Simple*' in SMTP. In this example we hook up directly to the receiving mail server playing the sending mail

server ourself.

At first we initiate the communication on the port 25 to the server:

telnet mymailserver.ch 25

The receiving mail server will now take the connection and replies with a couple of informations like what mail server is running, what escape sequence is accepted and sometimes even informations about what mail services are available. These informations depend a lot on the MTA used. Now because we are a polite mail server we greet the other mail sever first. For greeting we specify the domain we are currently in so the mail server can reject us already if we would be a bad spammer trying to flood it. Now let's greet:

HELO my.domain.ch

The mail server is happy and sends us a *250* message indicating beeing happy. Now we tell first who we are. It's like on the phone. We tell what mail adress wants to submit a mail. This is done like this:

MAIL FROM: <user@domain.ch>

The brackets are important and have to be there. At this stage the MTA can check if a certain mail adress is blocked or otherwise disallowed. Spam filters are common to step in at this point to stop spam. These are usually hard code filters set by the Mail Provider and can not be modified by the users. This can sometimes lead to certain frustration about not reaceived mails. After specifiying who send the mail we have to tell where the mail shall end up. This is done like this:

RCPT TO: <user@domain.ch>

Again the brackets are mandatory and have to be there. Also spaces have to be correct. The mail server will now check if the given user exists. If this is ok we receive again a *250*. If this should not be the case we will receive an error code. They are listed at the end of this document. If the receiver is confirmed all is ready for receiving the mail. Now we send the entire mail prepared in advance. This we start with the following command:

DATA

Now all text send is considered beeing part of the mail. A more profound description of the mail headers and data follows right after this topic. To end the data block we enter a single dot on a line. Just a dot and nothing else. After the server received this end signal it will deliever the mail to the mailbox and return us a *250* value. After we received this confirmation we close the connection using the quit command:

QUIT

This time we receive the code *221* indicating the end of the communcation. After this the connection will be closed by the mail server and the mail has arrived.

The Mail Header and other data send

The mail data block consist of the so called Mail Header and the data. Attachments also are part of the mail data and don't belong to the header. A header contains besides some important data also a lot of informations about the transfer itself not relevant to the user. The following table will list some header fields commonly used.

From *sender*	Contains the name of the mail sender. The format is not always the same but the most usual form is *"Name"* <*e-mail*>. Sometimes date informations are included there too. This field is displayed by the mail client as the sender. Unfortunatly this field can be falsified and thus is not reliable.
Return-path <*e-mail*>	This is the adress to reply to if needed. Again this field can be falsified and is often used to trap users because the mail suddenly gets send to another adress instead of the sender.
Delievered-to *infos*	Contains informations about the system having received the mail. This is optional and filled with some string of the mail server.

Received *infos*	Each time a mail gets transmitted from one MTA to another the MTAs involded usually prepend some stuff to the mail header. These informations can be used to follow the path a mail has taken until it arrived at the user.
Message-ID: *<id-adress>*	In some cases the sender mail server creates an id for each mail and it will be written at that place. This can be used for notification callbacks like the 'has-received' options available in certain mail clients like the ones mentioned above.
To: *adress*	Specifies the adress the mail was sent to. Can also be messed up which looks funny if the mail client delievers a mail to you that has another e-mail set as from where you got it.
Subject: *info*	The Subject line telling what this mail is all about.
Date: *info*	The date when the mail was sent. Another nice field to put funny stuff into like mails from the future.

Here an example of a possible mail header. The received lines have to be read bottom up.

From mustermann@hotmail.com Wed Aug 18 00:52:08 1999
Return-Path: <mustermann@hotmail.com>
X-Flags: 0001
Delivered-To: GMX delivery to paramind@gmx.net
Received: (qmail 5433 invoked by uid 0); 18 Aug 1999 00:52:08 -0000
Received: from f135.law3.hotmail.com (HELO hotmail.com) (209.185.241.135)
by mx6.gmx.net with SMTP; 18 Aug 1999 00:52:08 -0000
Received: (qmail 41386 invoked by uid 0); 18 Aug 1999 00:52:01 -0000
Message-ID: <19990818005201.41385.qmail@hotmail.com>
Received: from 195.252.134.115 by www.hotmail.com with HTTP;
Tue, 17 Aug 1999 17:52:01 PDT
X-Originating-IP: [195.252.134.115]
From: "Max Mustermann" <mustermann@hotmail.com>
To: paramind@gmx.net
Subject: Level2-Zugang
Date: Wed, 18 Aug 1999 02:52:01 CEST
Mime-Version: 1.0
Content-Type: text/plain; charset=iso-8859-1; format=flowed

Command and Code tables

There are more than only the mentioned commands you can use in a mail transfer and a couple of error codes you can get. Below are listed the most common ones used in mail transfer.

HELO <domain>	Begins the communication
MAIL FROM: <adresse>	Begins mail transfer giving the sender adress
RCPT TO: <adresse>	States the adress to deliver the mail to
DATA	Starts transmitting the mail data
RSET	Resets the current transmition
VRFY <adresse>	Checks if the given adress is valid on this server

EXPN <adresse>	Enumerates all mail distributors
HELP <befehl>	Gives help about a command
NOOP	Does nothing else than provoking a 250 response. Can be used to check if the mail server is still working
QUIT	Terminates transmition

The following error codes a returned commonly.

250	Success
251	User not local and has to be forwarded (adress will follow)
221	Closing transmition channel (after QUIT)
421	Service not ready. Happens if MTAs are suddenly down
450	Mailbox not available. Another error MTAs can occur if a user was removed
452	Out of Memory. Unlikely today but in old days was possible
500	Unvalid Command. Issued if you typed something wrong
502	Command not implemented. Should never happen
503	Bad Command Sequence. Occurs if you issued commands in the wrong order
554	Transaction Failed. The complete failure. Similar to a Stop Error on NT

Theorie Questions

Question	*Answer*
On what port a mail server is listening for incoming mail transfers?	On Port 25
What command does a mail client have to send to the mail server to initiate a *Mail* transfer?	*MAIL FROM: <mail-addr>* *Example: MAILFROM: <rptd.dnsalias.net>*
How does a mail client mark the end of a data block?	A single dot on an otherwise empty line
On what way does a mail usually travel towards it's destination if you don't use a telnet application but a conventional mail client?	*The mail client sends the mail first to its relaying mail server. This mail server then sends the mail to the receiving mail server where the mails ends in the mailbox of the user.*
What Code signals a succesfull processing of a command by the mail server?	The Code 250

Which source adress is trustfull and why?	The mail adress in the *MAIL FROM* command is trustfull because the other mail adress in the *DATA* block can be manipulated very easily.