In this document we provide some guidelines for testing the `conf` multimedia application. For all tests we expect audio quality to be high.

Perform all tests with 'verbose' mode set to on, and check the behavior of sent and received packets. For all cases, identify which should be the appropriate behavior, and observe if this behavior result from your tests?

Audio formats. Check communication for both audio formats.

Accurate timing. Identify all the events in your code should be performed in a regular fashion (i.e. periodically). Check if this true using the `diffTime` which you can find in the code repository. This tool estimates the time difference between two events shown by `strace`, to ease your validation of the periodicity of your code.
Assume, for example, that you think that the event of writing to a given socket/file descriptor (let's say file descriptor number 5) should be periodic. To check this assumption, you should execute

```
strace –tt –o confTrace ./conf  …
cat confTrace | grep 'write\(5' | ./diffTime
```

`-tt` request `strace` to show the time at which each system call is performed
The `grep` command is used to select which events are relevant, and remove any non-relevant events in the trace provided by `strace`
Finally `diffTime` computes the time between two events (instead of using absolute time values)

You can check multiple events by using 'grep –E'. For example, you can filter out all lines except those containing 'select', 'write(4' and 'read(4' by executing

```
cat confTrace | grep –E 'select|write\(4|read\(4' | ./diffTime
```

grep –E enables the use of regular expressions. Note that characters such as '`(`' must be preceded by the escape character '`\`'.

Be sure that your code only blocks at `select`. Check that the time between two consecutive reproductions (recordings) of packets is roughly equal to the packet duration

Buffering. Is buffering working? Use large buffering values, in the order of seconds, and check if the communication is actually delayed for that period.
How does your application behave when buffering is set to 0? Is this normal?

Packet size. The application should work for different packet sizes. Small packet sizes may pose some problems for inefficient applications. Good quality should be achieved at least starting with 64-bytes packets.

RTP (and RTCP format). The best way to check if RTP (and eventually RTCP) messages have been formed correctly is to use `rtpdump` (see http://www.cs.columbia.edu/~hgs/rtptools, installed in the labs), a third party tool. To use it, you need THREE different systems (either three physical computers, or three virtual ones). It will not work in just two.
Let's name them H1, H2, H3, and consider MCAST_ADDR a valid multicast address:

```
H1:~> ./conf first -mMCAST_ADDR

H2:~> rtpdump MCAST_ADDR/5004

H3:~> ./conf second MCAST_ADDR
```

In this case, H3 is sending packets to the multicast address to which H1 and H2 are listening. H1 uses unicast to send its packets to H3.

You should check that

- `rtpdump` identifies the packet format, similar to the following:

```
1372261892.985270 RTP len=140 from=163.117.144.133:5004 v=2 p=0 x=0 cc=0 m=0
pt=100 (????,0,0) seq=0 ts=0 ssrc=0x0

1372261893.001268 RTP len=140 from=163.117.144.133:5004 v=2 p=0 x=0 cc=0 m=0
pt=100 (????,0,0) seq=1 ts=128 ssrc=0x0
```

An example of BAD format identification (i.e., there is a problem with the way the packet is built) is

```
1372261893.066233 VATD len=140 from=163.117.144.133:5004 nsid=0 flags=0x97
confid=4 ts=3077347072
```

- The sequence number increases 1 by 1
- The timestamp value increases appropriately (i.e., in the number of samples contained in every packet)

Lost packets. Check that your application behaves properly when packets are lost. To check this, in the part of your code that sends to the other party, insert code changing the next sequence number from 50 to 52 (meaning that your sending code will send a packet numbered 48, 49, then 52, 53, 54, etc.). Check that the code copies twice (and reproduces three times) packet number 49.