Experimenting with a LMAP testbed

by Vaibhav Bajpai `v.bajpai@jacobs-university.de`

A measurement system within a Large Scale Measurement of Access network Performance (LMAP) ecosystem consists of a number of strategically located Measurement Agents (MA)s, a controller and a federation of measurement data collectors. A MA in such a measurement system starts by bootstrapping a registration process with a controller. The controller on successfully identifying the MA, sends instructions on which measurement tests are to run, along with a test schedule that defines when and with what frequency they are to be run using a LMAP control protocol. The MA after running the tests reports the test results to a collector using a LMAP report protocol. The data stored on the collector can later be retrieved for further analysis.

This document prepares an experimental LMAP testbed. It consists of a MA, a controller and a collector. For simplicity reasons, the functionality of the controller and collector have been merged into a single unit. The document identifies how a Representational State Transfer (REST) architectural style with HTTP as the transport protocol can be used both as a control and a report protocol in the LMAP ecosystem in practice. This document does not necessarily represent how the control and report protocol is going to be standardized by the IETF. The API described here is only to ascertain the feasibility of the REST architectural style with HTTP as a transport medium for use as the control and report protocols.

The instructions given below are assumed to be run on a virtualized CPE, prepared in the previous handout. This is because the binaries used in the testbed are cross-compiled for the mips architecture.

Bootstrap the virtual CPE as a LMAP MA with the controller using HTTP REST

Any interaction with the controller requires that the MA must first know the URI location of the controller. This information may be sent during the session establishment via TR-069, but is outside the scope of this tutorial. Assuming the CPE already knows the URI location of the controller, test the connectivity with the controller:

```
$ export SERVER=cook.eecs.jacobs-university.de:5000
$ opkg install curl
$ curl -X GET $SERVER
hello world
```

A MA in LMAP is identified using a UUID. A MA may not have the capability to generate a UUID locally, therefore it must obtain this UUID from the controller during the bootstrap process.

On bootstrap, the MA must retrieve a MA UUID from the controller by sending its MAC address. Ideally this information must be exchanged within a two-way authenticated TLS encrypted session. For the sake of simplicity, we have kept the security discussion out of the context of the tutorial. The virtual CPE, however, may not have a unique MAC address for each participant, so we will use a random unique string to avoid conflicts. Retrieve a random UUID to treat it as a unique MAC address string for your CPE:

```
$ export MAC=`curl -X GET $SERVER/uuid`
$ echo $MAC
c824ac2dce411e282ef00e08147c934
```

The virtual CPE then sends a HTTP POST request to register as a LMAP MA. Given the MAC address, the server creates a measurement point and returns the measurement ID. If the MAC address is not sent, server sends a 400 (bad request) response. If the MAC address already exists in database, server sends the existing measurement ID. The data to be sent must be encoded in the JSON format.

```
$ export UUID=`curl -X POST -H "Content-Type: application/json" -d '{ "mac" : "$MAC" }'  $S
ERVER/msmpoint`
$ echo $UUID
60bd618edce511e282ef00e08147c934
```

The controller on receiving the MAC address creates a persistent context, generates a UUID and sends it back to the MA. If the collector has already seen the MAC address before, it already has the context and just sends the previously created measurement ID.

Install a measurement test

In a LMAP ecosystem, the test binaries shall be pushed by the controller to the MA as part of the regular software updates. For the sake of this testbed, we will ourselves retrieve a test binary.

```
$ export MTRDIR=`pwd`
$ mkdir $MTRDIR/bin
$ curl cnds.eecs.jacobs-university.de/users/vbajpai/aims-tutorial-2013/mtr > $MTRDIR/bin/mtr
```

Test if the binary works:

```
$ chmod +x $MTRDIR/bin/mtr
$ $MTRDIR/bin/mtr -h
usage: ./mtr [-hvrwctglspniuT46] [--help] [--version] [--report]
                [--report-wide] [--report-cycles=COUNT] [--curses] [--gtk]
                [--csv|-C] [--raw] [--split] [--mpls] [--no-dns] [--show-ips]
                [--address interface] [--filename=FILE|-F]
                [--ipinfo=item_no|-y item_no]
                [--aslookup|-z]
                [--psize=bytes/-s bytes]
                [--report-wide|-w] [-u|-T] [--port=PORT] [--timeout=SECONDS]
                [--interval=SECONDS] HOSTNAME
$ $MTRDIR/bin/mtr --csv 8.8.8.8
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;1;10.50.255.251;279
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;2;212.201.44.214;2435
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;3;vkr-g2-5-1.x-win.uni-bremen.de;826
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;4;xr-bre1-pc2.x-win.dfn.de;1982
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;5;cr-han1-te0-0-0-8.x-win.dfn.de;2871
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;6;cr-tub1-te0-7-0-4.x-win.dfn.de;7583
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;7;google.bcix.de;7882
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;8;209.85.249.184;38213
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;9;66.249.95.143;8367
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;10;64.233.174.55;8685
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;11;???;0
MTR.0.84+git:469d0eb3;1372090069;OK;8.8.8.8;12;google-public-dns-a.google.com;8643
```

Schedule the measurement test

Create a results directory to store the measurement results:

```
$ mkdir -p $MTRDIR/results
```

Create a file to store the results

```
$ touch $MTRDIR/results/$UUID
```

Create an input file with the following contents for the measurement test

```
$ vim $MTRDIR/topsites
www.google.com
www.facebook.com
www.youtube.com
www.yahoo.com
```

Create a script that runs the test and stores the result in the above file:

```
$ vim $MTRDIR/mtr.sh
#!/usr/bin/env sh

MTRDIR=$1
UUID=$2

$MTRDIR/bin/mtr -c 1 --no-dns -4 --csv --aslookup --filename \
$MTRDIR/topsites >> $MTRDIR/results/$UUID 2> /dev/null
```

Make the script executable:

```
$ chmod +x $MTRDIR/mtr.sh
```

Test the script:

```
$ sh $MTRDIR/mtr.sh $MTRDIR $UUID
$ cat $MTRDIR/results/$UUID

MTR.0.84+git:469d0eb3;1372092118;OK;www.facebook.com;1;10.50.255.251;AS???;267
MTR.0.84+git:469d0eb3;1372092118;OK;www.facebook.com;2;212.201.44.214;AS680;718
MTR.0.84+git:469d0eb3;1372092118;OK;www.facebook.com;3;134.102.121.125;AS680;815
MTR.0.84+git:469d0eb3;1372092118;OK;www.facebook.com;4;188.1.239.205;AS680;1228
...
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;1;10.50.255.251;AS???;273
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;2;212.201.44.214;AS680;1102
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;3;134.102.121.125;AS680;824
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;4;188.1.239.205;AS680;1150
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;5;188.1.145.241;AS680;2958
MTR.0.84+git:469d0eb3;1372092118;OK;www.google.com;6;188.1.145.217;AS680;7476
...
```

Create a test running schedule using `cron`. The following commands fetch the current cron entries in `currentcron` file and appends an additional cron entry for the measurement test schedule. The cron entry runs the `mtr.sh` script every 5th minute of the hour. The `currentcron` file is later patched back into

the system.

```
$ crontab -l > currentcron 2> /dev/null
$ echo "5 * * * * $MTRDIR/mtr.sh \"$MTRDIR\" \"$UUID\"" >> currentcron
$ crontab currentcron
$ rm currentcron
```

Send the measurement test results to a collector using HTTP REST

Send a HTTP POST request to the collector:

```
$ curl -X POST -F results=@"$MTRDIR/results/$UUID" "$SERVER/mtr/$UUID/result"
  60/60 rows inserted successfully
```

Delete the results file:

```
$ rm -rf $MTRDIR/results/$UUID
```

Schedule the reporting of measurement test results

Create a script that sends the results to the collector and then deletes the results locally.

```
$ vim $MTRDIR/submit.sh
#!/usr/bin/env sh

MTRDIR=$1
UUID=$2
SERVER=$3

curl -X POST -F results=@"$MTRDIR/results/$UUID" "$SERVER/mtr/$UUID/result" > /dev/null
2>&1 && \
rm -rf $MTRDIR/results/$UUID
```

Make the script executable:

```
$ chmod +x $MTRDIR/submit.sh
```

Test the script

```
$ sh $MTRDIR/submit.sh $MTRDIR $UUID $SERVER
```

Create a result reporting schedule using `cron`: The following commands fetch the current cron entries in `currentcron` file and appends an additional cron entry for the measurement test schedule. The cron entry runs the `submit.sh` script every 45th minute of the hour. The `currentcron` file is later patched back into the system.

```
$ crontab -l > currentcron 2> /dev/null
$ echo "45 * * * * $MTRDIR/submit.sh \"$MTRDIR\" \"$UUID\" \"$SERVER\"" >> currentcron
$ crontab currentcron
$ rm currentcron
```

Check the final cron schedule

```
$ crontab -l
```

Retrieve the measurement test results back from the collector using HTTP REST

Send a HTTP GET request to retrieve the measurement result metadata of a test, <test> collected from the measurement point with measurement ID, <msmid>. If the measurement ID does not exist in the given test table, the server sends a 404 (not found) response. The metadata response is encoded in the JSON format.

```
$ curl -X GET $SERVER/mtr/$UUID
  {
      "start": 1372092118
      "end": 1372093784,
      "result": "/mtr/2904e204dce911e282ef00e08147c934/result",
  }
```

Send a HTTP GET request to the collector to retreive the measurement results of a test, <test> collected from the measurement point with measurement ID, <msmid>. If the measurement ID does not exist in the given test table, the server sends a 404 (not found) response. The measurement results are encoded in the JSON format.

```
$ opkg install python
$ curl -X GET $SERVER/mtr/$UUID/result | python -m json.tool | less
[
  {
      "asn": "AS???",
      "endpoint": "10.50.255.251",
      "msmid": "2904e204dce911e282ef00e08147c934",
      "service": "www.youtube.com",
      "status": "OK",
      "time": 285,
      "timestamp": 1372092118,
      "ttl": 1,
      "version": "MTR.0.84+git:469d0eb3"
  },
  {
      "asn": "AS680",
      "endpoint": "212.201.44.214",
      "msmid": "2904e204dce911e282ef00e08147c934",
      "service": "www.youtube.com",
      "status": "OK",
      "time": 713,
      "timestamp": 1372092118,
      "ttl": 2,
      "version": "MTR.0.84+git:469d0eb3"
  },
  ...
 ]
```

Appendix

A schema to maintain the context of a measurement point is shown below. A measurement point is identified by a measurement ID, <msmid> and must contain a unique MAC address. The MAC address is sent my the measurement agent when it bootstraps and registers for the first time by sending a HTTPS POST request to /msmpoint.

```
create table msmpoint (

   msmid         TEXT        PRIMARY KEY
 , mac           TEXT        NOT NULL    UNIQUE
 , asn           TEXT
 , name          TEXT
 , location      TEXT
 , email         TEXT

);
```