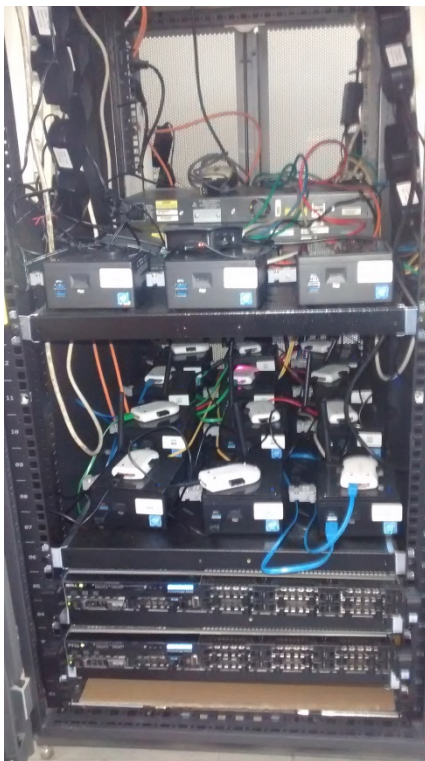


IEEE Conference on Network Function Virtualization and
Software Defined Networks
6-8 November 2017 – Berlin, Germany

Tutorial on unikernels, Edge Computing & IoT

Hands-on Paper



The experiments will run on emulab infrastructure based on the department of Applied Informatics of University of Macedonia, Thessaloniki, Greece.

There are nine machines dedicated to the current tutorial. They all have an IoT (zolertia zoul) device connected via the USB (serial-tunslip).

Node 1: Temperature + Humidity, Gyroscope
Node 2: Temperature + Humidity, Gyroscope
Node 3: Temperature + Humidity
Node 4: Light
Node 6: Light
Node 7: Light
Node 8: Barometer
Node 9: Barometer
Node 11: Barometer
Node 13: Gyroscope

Figure 1: miniPCs with IoT Rack in UoM, Greece

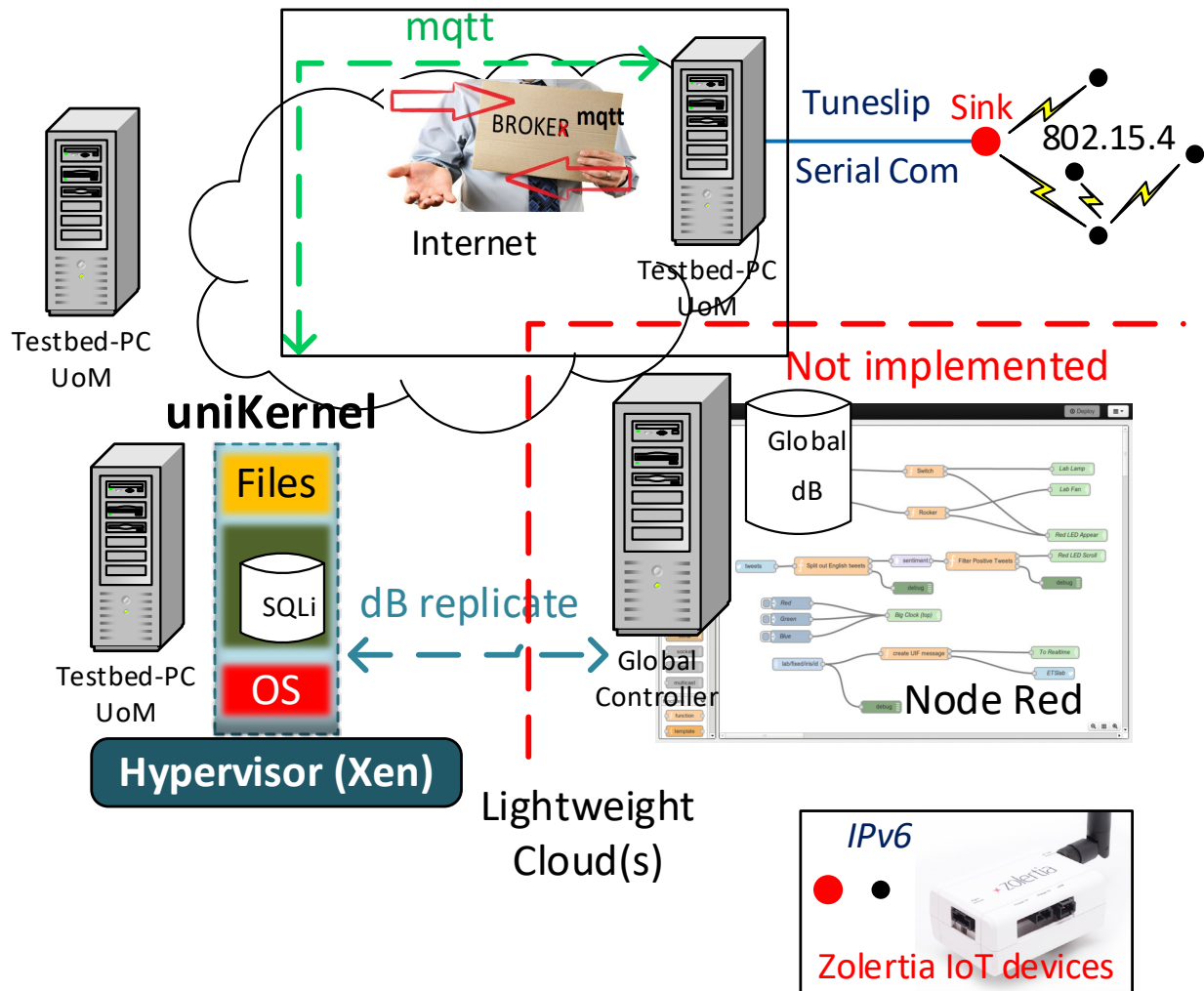


Figure 2: Abstract View of the Experiment

In general, you can copy & paste all code lines (font: courier)

Hands-on 1.1 IoT – Contiki

Read Sensor Data from an IoT Device

Login to one of the following stations (Use putty or ssh): `mpcXX.swn.uom.gr`

mpc1	mpc2	mpc3	mpc4	mpc6	mpc7	mpc8	mpc9	mpc11	mpc13
------	------	------	------	------	------	------	------	-------	-------

Username: `nfvsdn`

Password: `nfv123`

Serial Port communication

For the experiment to work, we need the current user to be able to communicate with the serial port. The appropriate rights must be delegated to the user `nfvsdn`

To check, run the command:

```
groups
```

You should see

```
nfvsdn2017 root dialout
```



```
nfvsdn@node7: ~  
nfvsdn@node7:~$ groups  
nfvsdn2017 root dialout  
nfvsdn@node7:~$
```

If you don't see `dialout`, do the following:

```
sudo usermod -a -G dialout nfvsdn # nfvsdn is the username
```

Then **you must close the terminal and login again**

Go to the example folder:

```
cd ~/contiki/examples/swin/01-iotsense/
```

Declare the target device, i.e. the type of IoT device (zoul = zolertia)

```
make TARGET=zoul savetarget
```

Compile the particular code file (iotsense.c). Be careful, no `."` at the end.

```
make iotsense
```

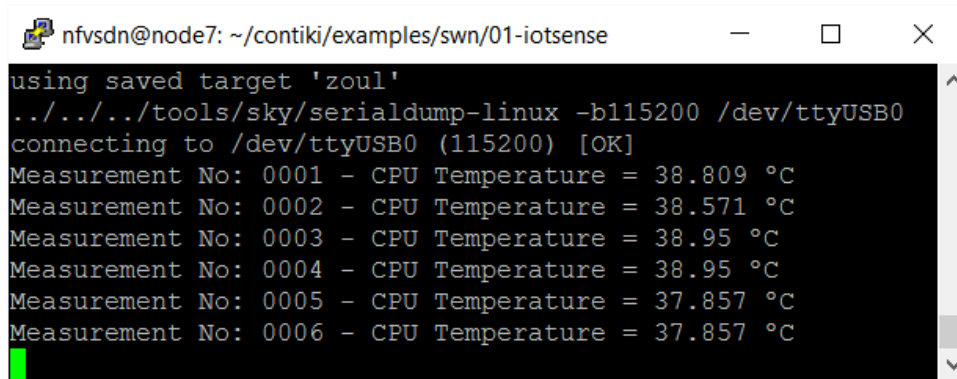
Upload to the IoT device the already compiled file

```
make iotsense.upload
```

Login into the IoT device (zolertia connected via USB/serial) and see the results of the above code:

```
make login
```

You should see the following:



```
nfvsdn@node7: ~/contiki/examples/swin/01-iotsense
using saved target 'zoul'
../../../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB0
connecting to /dev/ttyUSB0 (115200) [OK]
Measurement No: 0001 - CPU Temperature = 38.809 °C
Measurement No: 0002 - CPU Temperature = 38.571 °C
Measurement No: 0003 - CPU Temperature = 38.95 °C
Measurement No: 0004 - CPU Temperature = 38.95 °C
Measurement No: 0005 - CPU Temperature = 37.857 °C
Measurement No: 0006 - CPU Temperature = 37.857 °C
```

Summary: You compiled a C language code, uploaded it to the IoT device, and did login to this device to see the results (the IoT device “sees” your terminal as an output-terminal device).

Hands-on 1.2 (a) IPv6 – Broadcast

IPv6 Broadcasting of IoT sensor data

Login to one of the following stations (Use putty or ssh): `mpcXX.swn.uom.gr`

PC	mpc1	mpc2	mpc3	mpc4	mpc6	mpc7	mpc8	mpc9	mpc11	mpc13
IP	203	204	205	206	208	209	210	211	213	215
Emulab	Node8	Node7	Node2	Node10	Node5	Node3	Node9	Node6	Node1	Node4
IPv6	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0212	0212	0212	0212	0212	0212	0212	0212	0212	0212
	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00
	060d	060d	060d	060d	060d	060d	060d	060d	060d	060d
b123	60e2	b152	6227	b13c	6145	615e	6132	60b1	60b5	

Username: `nfvsdn`

Password: `nfv123`

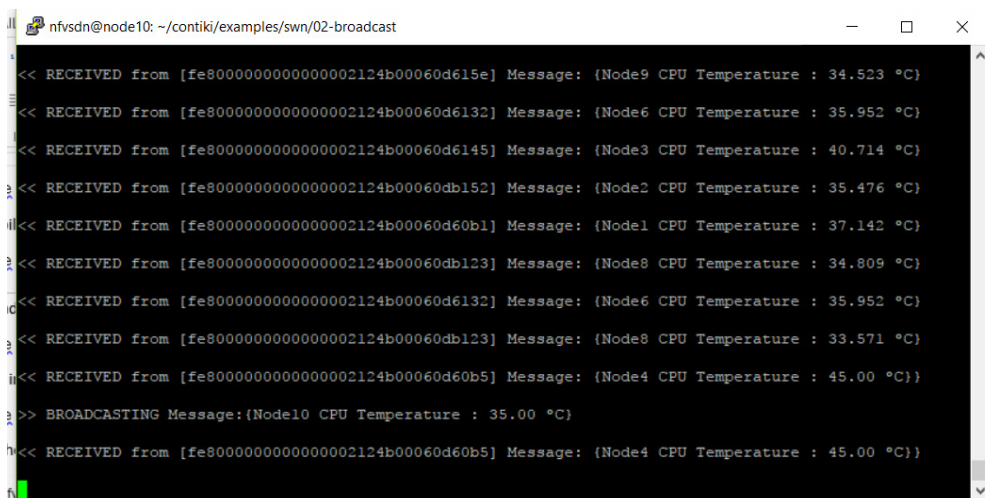
```
cd ~/contiki/examples/swn/02-broadcast
```

Compile and Upload to the IoT device

```
make broadcast.upload
```

Login into the IoT device (zolertia connected via USB/serial) and see the results of the above code:

```
make login
```



```

nfvsdn@node10: ~/contiki/examples/swn/02-broadcast
<< RECEIVED from [fe8000000000000002124b00060d615e] Message: {Node9 CPU Temperature : 34.523 °C}
<< RECEIVED from [fe800000000000000002124b00060d6132] Message: {Node6 CPU Temperature : 35.952 °C}
<< RECEIVED from [fe800000000000000002124b00060d6145] Message: {Node3 CPU Temperature : 40.714 °C}
<< RECEIVED from [fe800000000000000002124b00060db152] Message: {Node2 CPU Temperature : 35.476 °C}
<< RECEIVED from [fe800000000000000002124b00060d60b1] Message: {Node1 CPU Temperature : 37.142 °C}
<< RECEIVED from [fe800000000000000002124b00060db123] Message: {Node8 CPU Temperature : 34.809 °C}
<< RECEIVED from [fe800000000000000002124b00060d6132] Message: {Node6 CPU Temperature : 35.952 °C}
<< RECEIVED from [fe800000000000000002124b00060db123] Message: {Node8 CPU Temperature : 33.571 °C}
<< RECEIVED from [fe800000000000000002124b00060d60b5] Message: {Node4 CPU Temperature : 45.00 °C}
>> BROADCASTING Message:{Node10 CPU Temperature : 35.00 °C}
<< RECEIVED from [fe800000000000000002124b00060d60b5] Message: {Node4 CPU Temperature : 45.00 °C}

```

Hands-on 1.2 (b) IPv6 – Multi-hop RPL -UDP

IPv6 Multi-hop RPL UDP

Username: nfvsdn

Password: nfvl23

```
cd ~/contiki/examples/swn/03-multihop
```

	UDP-Server	UDP-Clients								
PC	mpc1	mpc2	mpc3	mpc4	mpc6	mpc7	mpc8	mpc9	mpc11	mpc13
IP	203	204	205	206	208	209	210	211	213	215
Emulab	Node8	Node7	Node2	Node10	Node5	Node3	Node9	Node6	Node1	Node4
IPv6	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80	fe80
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
	0212	0212	0212	0212	0212	0212	0212	0212	0212	0212
	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00	4b00
	060d	060d	060d	060d	060d	060d	060d	060d	060d	060d
	b123	60e2	b152	6227	b13c	6145	615e	6132	60b1	60b5

Server Side – Tutor MPC1 – (node 8) : udp-server.c

```
make udp-server.upload login
```

Client Side – Students' PCs (MPC2,3,4,6,7,8,9,11,13)

```
make udp-client.upload login
```

Exercises:

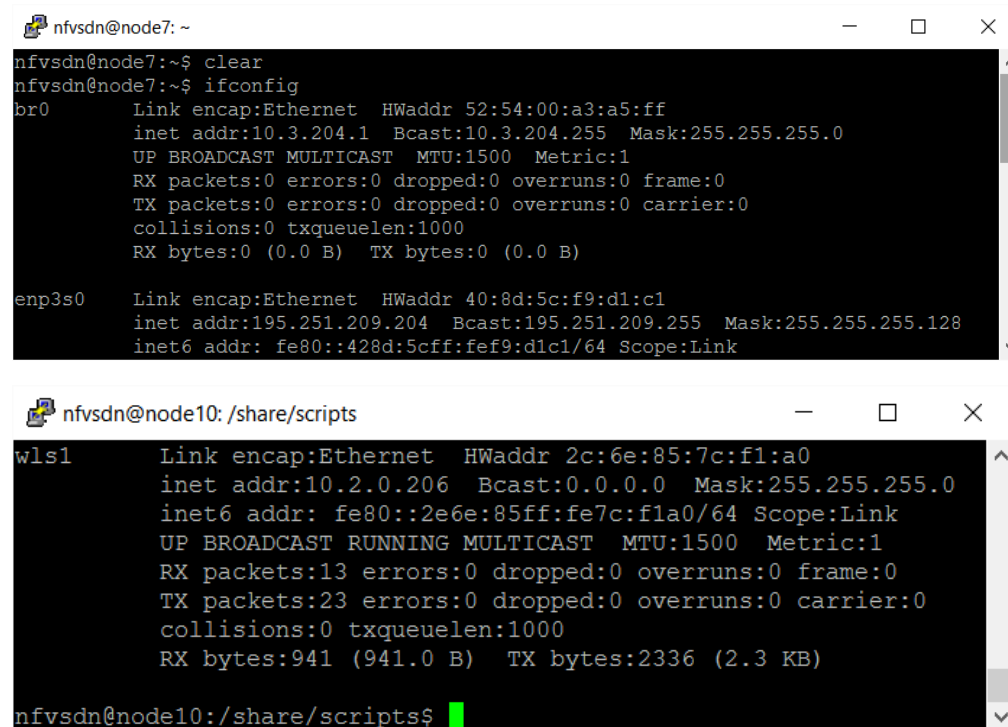
1. Decrease the message frequency...
2. Change the message from hello to your name...
3. Send the CPU temperature to the sink...

Hands-on 2.1 Unikernels

In this example, we will run a unikernel which is loading a dB (sqlite), connects with the sink IoT (zolertia) connected to the USB port (emulated as a serial via tuneslip) of mpc1, and then subscribes to an mqtt broker to receive information from all IoT devices-clients connected to the sink.

Run the command `ifconfig`

If you can see the interface `br0` AND THE INTERFACE `wsl1` HAVING AN IPv4 address, omit this step.



```
nfvdsn@node7: ~  
nfvdsn@node7:~$ clear  
nfvdsn@node7:~$ ifconfig  
br0    Link encap:Ethernet  HWaddr 52:54:00:a3:a5:ff  
       inet addr:10.3.204.1  Bcast:10.3.204.255  Mask:255.255.255.0  
       UP BROADCAST MULTICAST  MTU:1500  Metric:1  
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
       TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
       collisions:0 txqueuelen:1000  
       RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
  
enp3s0  Link encap:Ethernet  HWaddr 40:8d:5c:f9:d1:c1  
       inet addr:195.251.209.204  Bcast:195.251.209.255  Mask:255.255.255.128  
       inet6 addr: fe80::428d:5cff:fef9:d1c1/64 Scope:Link  
  
nfvdsn@node10: /share/scripts  
wsl1   Link encap:Ethernet  HWaddr 2c:6e:85:7c:f1:a0  
       inet addr:10.2.0.206  Bcast:0.0.0.0  Mask:255.255.255.0  
       inet6 addr: fe80::2e6e:85ff:fe7c:f1a0/64 Scope:Link  
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
       RX packets:13 errors:0 dropped:0 overruns:0 frame:0  
       TX packets:23 errors:0 dropped:0 overruns:0 carrier:0  
       collisions:0 txqueuelen:1000  
       RX bytes:941 (941.0 B)  TX bytes:2336 (2.3 KB)  
  
nfvdsn@node10:/share/scripts$
```

Otherwise, you must (re-) initialize the xen server parameters (dhcp server, ad-hoc networking, etc.):

```
cd /share/scripts  
./enable-ad-hoc  
./createmecnetwork  
./configure-network
```

Go inside the folder `rumprun-packages/python3/hello-world`

```
cd ~/rumprun-packages/python3/hello-world
```

Inside you will find one python file: `main.py`

If you don't want to change it (it is just the "hello-world" app), continue.

We need to create the main.iso HDD. In other words, this is going to be the primary HDD of the rumpkernel. It will be combined (and compiled) with all the necessary files for python libraries. This python libraries ISO has been created before (It takes a long time, and there is no need to re-compile it every time)

```
sudo genisoimage -r -o main.iso main.py tutorial-  
env/lib/python3.5/site-packages
```

```
cd ~/rumprun-packages/python3/
```

To run the unikernel make sure you are inside the folder:

```
~/rumprun-examples/python3/
```

Because of DHCP taxonomy, **BEFORE RUNNING the following, change the mac address** in line two according to the following: (IP: 195.251.209.XXX)

mpc1	mpc2	mpc3	mpc4	mpc6	mpc7	mpc8	mpc9	mpc11	mpc13
203	204	205	206	208	209	210	211	213	215
node8	Node7	Node2	Node10	Node5	Node3	Node9	Node6	Node1	Node4
CB	CC	CD	CE	D0	D1	D2	D3	D5	D7

Check the mac address second tuple **ABOVE**, and insert it in the second line below.

```
sudo /users/nfvdsn/rumprun/rumprun/bin/rumprun xen \  
-I if,xenif,'bridge=br0,mac=52:D5:00:00:00:2' \  
-W if,inet,dhcp -i \  
-b images/python.iso,/python/lib/python3.5 \  
-b hello-world/main.iso,/python/lib/python3.5/site-packages \  
-e PYTHONHOME=/python \  
-- examples/python.bin -m main
```

Analysis of the above commands:

```
sudo /users/nfvdsn/rumprun/rumprun/bin/rumprun xen \  
is running the executable of the rumpkernels (rumprun) specifically for xen VM.  
-I if,xenif,'bridge=br0,mac=52:CB:00:00:00:2' \  
-W if,inet,dhcp -i \  

```


Gets a DHCP address

```
-b images/python.iso, /python/lib/python3.5 \
```

The image python.iso is already created. It takes long time to create it since it is compiling all the libraries of python (think about it like the “Program Files”)

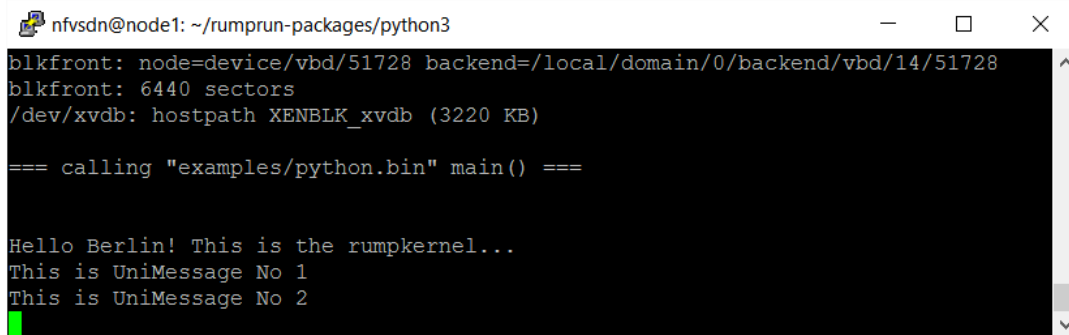
```
-b hello-world/main.iso, /python/lib/python3.5/site-packages \
```

Is creating another iso with all the user created files. Every time the user changes something (e.g. edit the main.py code, this will be recompiled)

```
-- examples/python.bin -m main
```

Is just indicating to the OS to find and execute the file named main(.py) (python file).

If all goes well, you must see messages printing!



```
nfvsn@node1: ~/rumprun-packages/python3
blkfront: node=device/vbd/51728 backend=/local/domain/0/backend/vbd/14/51728
blkfront: 6440 sectors
/dev/xvdb: hostpath XENBLK_xvdb (3220 KB)
=== calling "examples/python.bin" main() ===
Hello Berlin! This is the rumpkernel...
This is UniMessage No 1
This is UniMessage No 2
```

How to see/kill a running unikernel

To see if a unikernel is running (You need to open a **NEW terminal**):

```
sudo xl list
```

You see a list like this

Name	ID	Mem	VCPUs	State	Time (s)
Domain-0	0	3843	2	r-----	663.2
rumprun-python.bin	2	64	1	---s--	0.3

To destroy a particular unikernel:

```
sudo xl destroy [ID]
```

Hands-on 2.2 Lightweight-edge computing-IoT

Mosquito broker will already be running in mpc1. The following is information ONLY.

To run the mosquito broker (with “nohup” it runs on the background, so the terminal can be used for other activities). Make sure you include the ampersand (&) at the end:

```
nohup mosquitto &
```

press `ctrl+C` # mosquito will still run

if you want to kill mosquito:

```
killall mosquitto
```

Running IoT server-client

INFORMATION ONLY. NO NEED to run this

Running the border router

(For the experiment, the sink will be running only on one machine, i.e. mpc1)

```
cd ~/contiki/examples/swn/00-br6  
make border-router.upload && make connect-router
```

User needs to do the following:

This must be run to all available machines

Running the IoT client on **ANOTHER** machine than mpc1:

```
cd ~/contiki/examples/swn/04-mqtt-client  
make mqtt-client.upload
```

Running mqtt-connection unikernel example

```
cd ~/rumprun-packages/python3/examples/mqtt
```

There is a collection of files & programs there that we will include to the unikernel. File `main.py` is the starting point. There is also an sqlite python program (`sqli_mem2.py`) that creates a dB and stores the data from the IoT device. Keep in mind that all the client IoT devices connected to the base (sink) device, will publish their data to the mqtt broker, which will be received via the USB port (tuneslip-serial) of mpc1. Each unikernel will subscribe to its "own" IoT device mqtt channel (`zolertia/evtnodeXX/status`) and then store the data inside the sqlite database in the unikernel.

The future works here are:

1. For the same unikernel to be transferred to another machine-server-infrastructure, while the mobile IoT device(s) is wandering around the premises (city, building, certain area, etc.).
2. For each cloud based infrastructure, to have its own mqtt server, dedicated to the particular IoT. In this scenario, the IoT data is only travelling to the **edge of the network**. In a second phase, the central application is asking specific aggressive questions to each distributed database (e.g. send me the average temperature for the last three days of your IoT).

Testing prerequisites for unikernel

First thing is to check if the code is running successfully:

```
cd ~/rumprun-packages/python3/examples/mqtt
```

TO DO:

Open the file `main.py` (use your preferable Linux editor)

```
nano main.py
```

Find the line:

```
MQTT_TOPIC_EVENT = "zolertia/evt/status"
```

Change the `evt` word to `evtnodeXX` where `XX` is the node number. E.g. if you are in node7, change it to:

```
MQTT_TOPIC_EVENT = "zolertia/evtnode07/status"
```

```
nfvdsn@node7: ~/rumprun-packages/python3/examples/mqtt
GNU nano 2.5.3 File: main.py Modified
console_handler.setFormatter(formatterConsole)
logfile_handler.setFormatter(formatter)

logger.addHandler(console_handler)
logger.addHandler(logfile_handler)

# Change accordingly to the MQTT Broker and topic you want to use
# In the example it would be either "test.mosquitto.org" or "fd0$
# running a mosquitto broker locally
# MQTT_URL = "fd00::1"
# MQTT_URL = "127.0.0.1"

MQTT_URL = "195.251.209.203"
#MQTT_URL = "mpc3.swin.uom.gr"

MQTT_TOPIC_EVENT = "zolertia/evtnode07/status"
MQTT_TOPIC_CMD = "zolertia/cmd/leds"

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Linter
```

Save and exit (Ctrl+X, then enter y, then enter, if you used nano)

Run the program to check if all is ok:

```
python ./main.py # or simply ./main.py
```

You should see at least the following message:

```
Starting main.py
```

Possible problems

If you don't see anything else, it means that: 1. Mqtt broker is not running, 2. There is no sink IoT connected with the correct image, 3. There is no IoT client device connected to the sink.

If you see a message similar to: paho library is missing

```
sudo pip3 install paho-mqtt
```

If no problems, you should get something like the following:

```
INFO - Connecting to 195.251.209.203
DEBUG - CONNECTED to 195.251.209.203
INFO - Keeping connection alive . . .
INFO - Connected to 195.251.209.XXX
```

```
DEBUG - Subscribed to zolertia/evtnodeXX/status
DEBUG - Subscribed to zolertia/cmd/leds
The last two lines, assure you that the mosquito broker is up & running
```

Summary: You just executed a python code into the mini-pc for testing. You DIDN'T run the unikernel yet.

Preparing the unikernel HDD (hdd.img)

As mentioned above, you are inside the `~/rumprun-packages/python3/examples/mqtt` folder. In here you have all the necessary files for the unikernel. After you alter those files, you need to insert them into the `hdd.img`:

```
cd ~/rumprun-packages/python3/examples/
```

First you need to mount it:

```
sudo mount hdd.img /mnt/iso
```

Now the hdd is ready for editing. Go into the folder

```
cd ~/rumprun-packages/python3/examples/mqtt
```

and copy all fresh files into the mounted hdd:

```
sudo cp -r . /mnt/iso
```

You just updated the hdd.iso. Now unmount it:

```
sudo umount /mnt/iso
```

```
cd ~/rumprun-packages/python3
```

Running unikernel

INFORMATION ONLY

If dhcp is not assigning IP, go to the root, `cd \` and run `cat /tmp/mecnetwork.xml`. Look at the second tuple and note it (e.g. **D0**). This is what you insert in the mac address below

Because of DHCP taxonomy, **BEFORE RUNNING the following, change the mac address** in line two according to the following: (IP: 195.251.209.XXX)

mpc1	mpc2	mpc3	mpc4	mpc6	mpc7	mpc8	mpc9	mpc11	mpc13
203	204	205	206	208	209	210	211	213	215
node8	Node7	Node2	Node10	Node5	Node3	Node9	Node6	Node1	Node4

CB	CC	CD	CE	D0	D1	D2	D3	D5	D7
----	----	----	----	----	----	----	----	----	----

Check the mac address second tuple **ABOVE**, and insert it in the second line below.

```
sudo /users/nfvdsn/rumprun/rumprun/bin/rumprun xen \  
-I if,xenif,'bridge=br0,mac=52:CD:00:00:00:2' \  
-W if,inet,dhcp -i \  
-b images/python.iso,/python/lib/python3.5 \  
-b examples/hdd.img,/python/lib/python3.5/site-packages \  
-e PYTHONHOME=/python \  
-- examples/python.bin -m main
```

Make sure you are in dir /python3 :

```
cd ~/rumprun-packages/python3
```

You are ready to run the unikernel with the NEW hdd.img. Execute the above, **AFTER YOU CHANGED THE TUPLE in yellow.**

Possible Problem

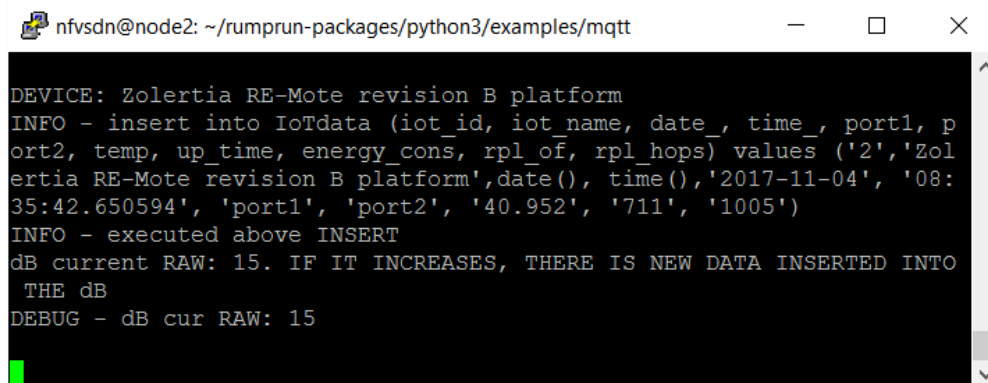
If the unikernel is stuck in this line:

```
INFO - Connecting to 10.2.0.203
```

OPEN A NEW terminal, and run the scripts again

```
cd /share/scripts  
./enable-ad-hoc  
./createmecnetwork  
./configure-network
```

You should see the unikernel receiving JSONs from the client IoT:



```
nfvdsn@node2: ~/rumprun-packages/python3/examples/mqtt  
DEVICE: Zolertia RE-Mote revision B platform  
INFO - insert into IoTdata (iot_id, iot_name, date_, time_, port1, p  
ort2, temp, up_time, energy_cons, rpl_of, rpl_hops) values ('2','Zol  
ertia RE-Mote revision B platform',date(), time(),'2017-11-04', '08:  
35:42.650594', 'port1', 'port2', '40.952', '711', '1005')  
INFO - executed above INSERT  
INFO - current RAW: 15. IF IT INCREASES, THERE IS NEW DATA INSERTED INTO  
THE dB  
DEBUG - dB cur RAW: 15
```

Future Work

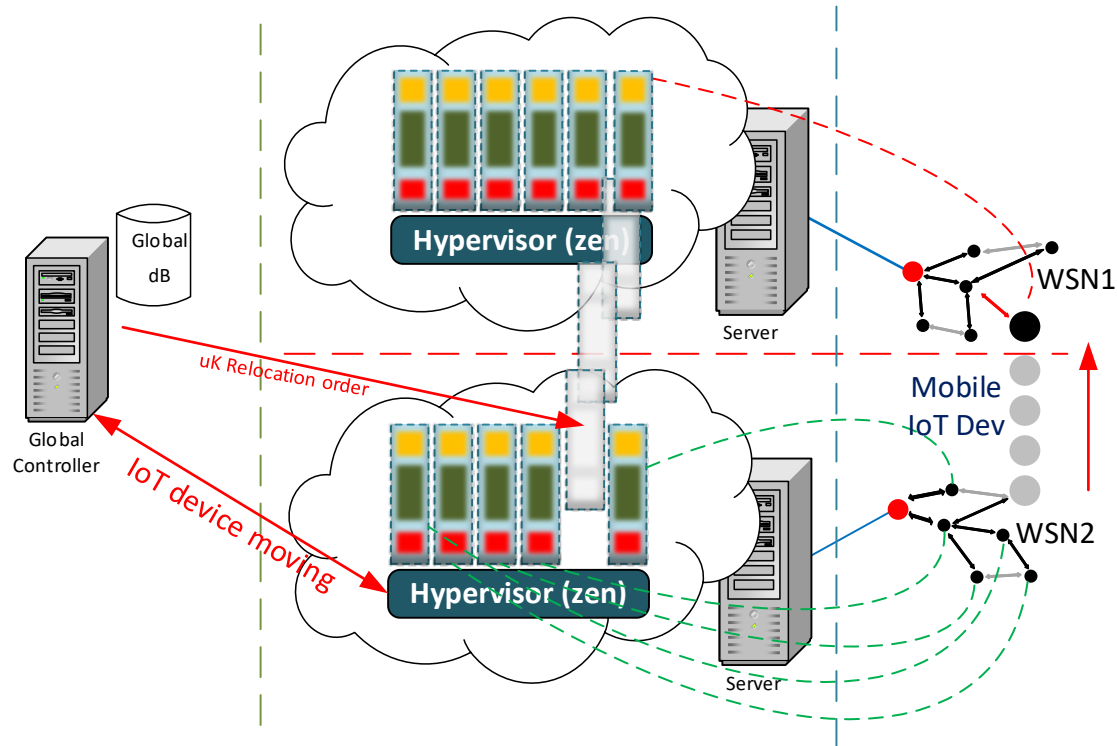


Figure 3: the unikernel is following the mobile IoT along the edge of the network for an implementation of Mobile Edge Computing (MEC)

Thank you
Lefteris Mamatas
George Violettas
Tryfon Theodorou