ToMaTo - Exercise

Overview

In this exercise we will use ToMaTo to develop and extend a **chat application**.

Exercise topics

- Simple topology management

Link emulation

Packet capturing

External networks

Programmable devices

Task overview

Topology creation

First chat test

External networks

Packet analysis

Enhancing the chat client

Link emulation

Chat monitor

Chat forwarder

Chat filter

Preparations

Working ToMaTo installation with an external network

Basic experience in networking, Linux and Python programming

A Linux computer with a browser with working java plugin

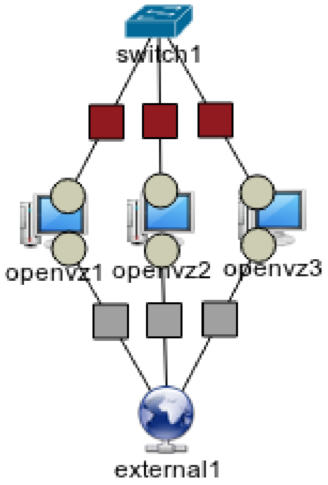
Experience with basic ToMaTo usage, Link emulation with ToMaTo and Repy programmable devices

Task 1: Topology creation

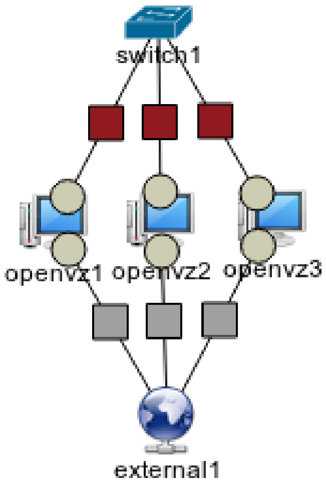
Details

1. Open the ToMaTo web frontend in your browser
2. Create a new topology with
   - Three OpenVZ nodes as chat clients
   - A switch connector that connects all chat client nodes
   - An external network that is connected to all chat client nodes

Hints



Solution of Task 1



Code: **`7rA`**

Task 2: First chat test

Details

1. Download the simple chat client
2. Login to the nodes using the console and start the openssh daemon
3. Upload the chat client to all client nodes
4. Run the chat client on all nodes and test the connection

Hints

- The IP addresses on the external network are assigned via DHCP, type **`ifconfig`** on the console of a node to find out its IP
- The openssh daemon can be started with the command **`/etc/init.d/ssh start`**
- Create an ssh key-pair using **`ssh-keygen`** and upload it to the nodes using **`ssh-copy-id`** to enable passwordless login.
- No python installed? Install it using **`sudo apt-get install python`**.
- The chat client expects the following parameters:
    1. The broadcast address: **`10.0.0.255`** in most cases
    2. The port to use: **`5000`**
    3. A nickname of the node: Be creative!

Task 3: External networks

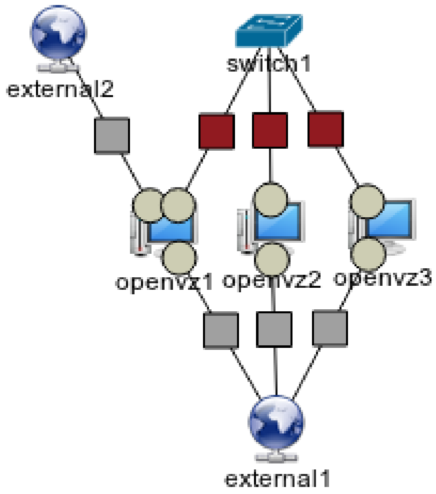
Details

1. Add another external network to your topology and connect one of your nodes to it
2. Configure the interface to use a fixed address in the range `192.168.0.0/16` **assigned to you**
3. Start the chat client again with broadcast address `192.168.255.255`
4. Can you chat with other people?

Hints

- Cannot connect the external network to device? You need to stop the device, connect it and start it again (Remember to start openssh daemon again)
- Use `ifconfig` to make sure your interface is configured correctly

Solution of Task 3



🔓 Code: **`Wm9`**

Task 4: Packet analysis

Details

1. Activate packet capturing on one of the switch connections
2. Look at the chat messages using a packet analysis tool
3. How big is a simple **`Hello world`** message?
4. Which protocol headers are included in the message and how much overhead do they cause?

Hints

- Edit the chat script to send a message periodically
- Cannot see any chat packets? Are you still chatting over the external network?
- If live capture does not work, use Cloudshark instead.

Solution of Task 4

Example packet

```
0000   ff ff ff ff ff ff 00 21  85 3c 4d 55 08 00 45 00   .......! .<MU..E.
0010   00 43 00 00 40 00 40 11  a2 73 83 f6 14 41 ff ff   .C..@.@. .s...A..
0020   ff ff 13 88 13 88 00 2f  3d 05 6d 73 67 3d 27 48   ......./ =.msg='H
0030   65 6c 6c 6f 20 77 6f 72  6c 64 27 3b 6e 69 63 6b   ello wor ld';nick
0040   6e 61 6d 65 3d 27 63 68  61 74 63 6c 69 65 6e 74   name='ch atclient
0050   27                                                 '
```

How big is a "Hello world message?

71 bytes + length of nickname, example nickname was "chatclient" so 81 bytes

Which protocol headers are included in the message and how much overhead do they cause?

Protocol	***Overhead***
Ethernet	*14 Bytes*
IPv4	*20 Bytes*
UDP	*8 Bytes*

Task 5: Enhancing the chat client

Details

- Change the chat client to enable link emulation tests in task 6
- Extend the chat client to include timestamp and increasing sequence number upon sending
- Determine reordering and delay upon receiving
  - Keep track of the next expected sequence number and report errors
  - Calculate delay as time difference
- Test your modifications in the topology

Hints

```python
# Map operations
map = {}
map["Bob"] = 6
map["Alice"] = 16
if "Bob" in map:
  print map["Bob"]
default_value = 0
print map.get("Charly", default_value)
```

```python
# Seconds since 1970-01-01, float value
timestamp = time.time()

# Float formatting
print "%.2f" % ( 0.5234235 * 10 )

# Assigning to global variables inside methods
var = 2
def method():
  global var
  var = 5
```

Solution of Task 5 (1/2)

New global variables

```
seqNum = 0
seqNums = {}
```

Changes to sending code

```
def send(msg): #Call this to send a message
  data = {"msg": msg, "nickname": nickname}
  data["time_sent"] = time.time()
  global seqNum
  seqNum += 1
  data["seqnum"] = seqNum
  sock.sendto(encode(data), address)
```

Solution of Task 5 (2/2)

Changes to receiving code

```python
def onReceive(src, msg): #This is called when new messages are received
  n = msg["nickname"]
  if n == nickname: #ignore messages from us
    return
  time_received = time.time()
  delay = time_received - msg["time_sent"]
  expected_seqnum = seqNums.get(n, 0)+1
  seqNums[n] = msg["seqnum"]
  print "From %(nickname)s: %(msg)s" % msg
  print "  Delay: %.2f ms" % ( delay * 1000.0 )
  if msg["seqnum"] != expected_seqnum:
    print "  Sequence number mismatch detected: seqnum=%d, expected=%d" % (msg["seqnum"], expected_seqnum)
```

Code: **b9A**

Task 6: Link emulation

Details

1. Upload the enhanced chat application from task 5 (Code: **`b9A`**) to the nodes
2. Change the link characteristics (e.g. delay, loss, duplication) of one connection and test the new features
3. Can you provoke packet reordering? (i.e. a new message is received after an older one)

Hints

- Let one client send a message periodically or try to type really fast
- Use big values for delay (up to several seconds) to see an effect
- Seeing negative delays? The host clocks might be out of sync

Solution of Task 6

Can you provoke packet reordering?

When the delay variance is high and the inter-packet time is small reordering can happen if the first message gets a high delay and the second message gets a low delay.

Task 7: Chat monitor

Details

1. Download the Repy library and unpack it
2. Create a Repy script that displays all chat messages that it receives
3. Add a programmable device to the topology, connect it to the switch
4. Upload your script to the programmable device and test the monitor

Hints

- Put your source file in **`src`**, call **`make`** and find your script in the **`build`** folder
- Use c-style **`#include <some/file>`** commands to include files from the library
- Have a look at **`ipmonitor.repy`**
- Make sure the headers exist before decoding them
- Use **`echo()`** instead of **`print`**

Solution of Task 7

Monitor code

```
#include <util/run_forever.repy>
#include <layer2/ethernet_proto.repy>
#include <layer3/ip_proto.repy>
#include <layer4/udp_proto.repy>

def monitor(src, pkt):
  eth = ethernet_decode(pkt)
  if eth.type != ETHERNET_TYPE_IP:
    return
  ip = ip_decode(eth.payload)
  if ip.protocol != IP_PROTOCOL_UDP:
    return
  udp = udp_decode(ip.payload)
  echo(udp.payload)

if callfunc == 'initialize':
    run_forever(monitor)
```

Code: **`F4f`**

Task 8: Chat forwarder

Details

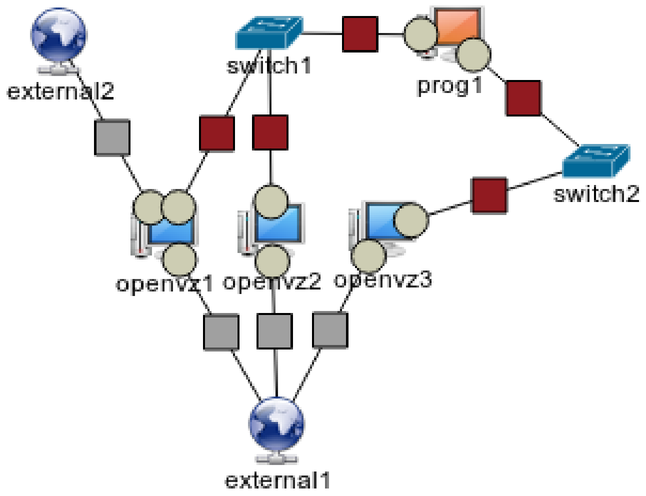
1. Write a Repy script that forwards chat messages to its interfaces
2. Adapt your topology so that you can test the forwarder
3. What happens if you connect two of these scripts to one switch?

Hints

- Only forward chat message packets
- Have a look at `switch.repy`
- The forwarder device needs two switches to forward
- Reconnect one OpenVZ node, so you can reuse it

Solution of Task 8 (1/2)

Adapted topology



What happens if you connect two of these scripts to one switch?

If the forwarder sends to the incoming interface, a ***forwarding loop*** *is the consequence. More complex topologies can even multiply messages and bring down the network. This problem can be solved with the spanning tree protocol.*

Solution of Task 8 (2/2)

Forwarder code

```
#include <util/run_forever.repy>
#include <layer2/ethernet_proto.repy>
#include <layer3/ip_proto.repy>
#include <layer4/udp_proto.repy>

def forward(src, pkt):
  eth = ethernet_decode(pkt)
  if eth.type != ETHERNET_TYPE_IP:
    return
  ip = ip_decode(eth.payload)
  if ip.protocol != IP_PROTOCOL_UDP:
    return
  udp = udp_decode(ip.payload)
  echo("Forwarding " + udp.payload)
  for dev in tuntap_list():
    if dev != src:
      tuntap_send(dev, pkt)

if callfunc == 'initialize':
    run_forever(forward)
```

Code: **9fF**

Task 9: Chat filter

Details

1. Extend the forwarder script (Code: **`9fF`**) to filter out some **bad words**
2. Read the list of bad words as parameters

Hints

```
# Extract arguments
args = callargs[0].split(",")

# Check substring
needle = "and"
haystring = "brand"
if needle in haystack:
  echo("found %s" % needle)
```

Solution of Task 9

Filtering forwarder code

```
#include <util/run_forever.repy>
#include <layer2/ethernet_proto.repy>
#include <layer3/ip_proto.repy>
#include <layer4/udp_proto.repy>

def forward(src, pkt):
  eth = ethernet_decode(pkt)
  if eth.type != ETHERNET_TYPE_IP:
    return
  ip = ip_decode(eth.payload)
  if ip.protocol != IP_PROTOCOL_UDP:
    return
  udp = udp_decode(ip.payload)
  for word in badwords:
    if word != "" and word in udp.payload:
      echo("Message filtered because of bad word: " + word)
      return
  echo("Forwarding " + udp.payload)
  for dev in tuntap_list():
    if dev != src:
      tuntap_send(dev, pkt)

badwords = callargs[0].split(",")

if callfunc == 'initialize':
    run_forever(forward)
```

Code: **gR7**

Summary

What have we learned?

- Basic ToMaTo usage
- Link emulation
- Packet capturing
- Programmable devices

Topics not covered

- Different operating systems using KVM
- Device image upload/download
- Device templates
- ToMaTo administration

Take-home tasks

- Write a chat robot that answers simple questions
- Change the chat client to allow unicast messages
- Write a chat server that keeps a list of participants and manages chat rooms
- Extend the chat client to forward messages to remote participants if needed
- Add acknowledgements to chat messages and implement retransmission