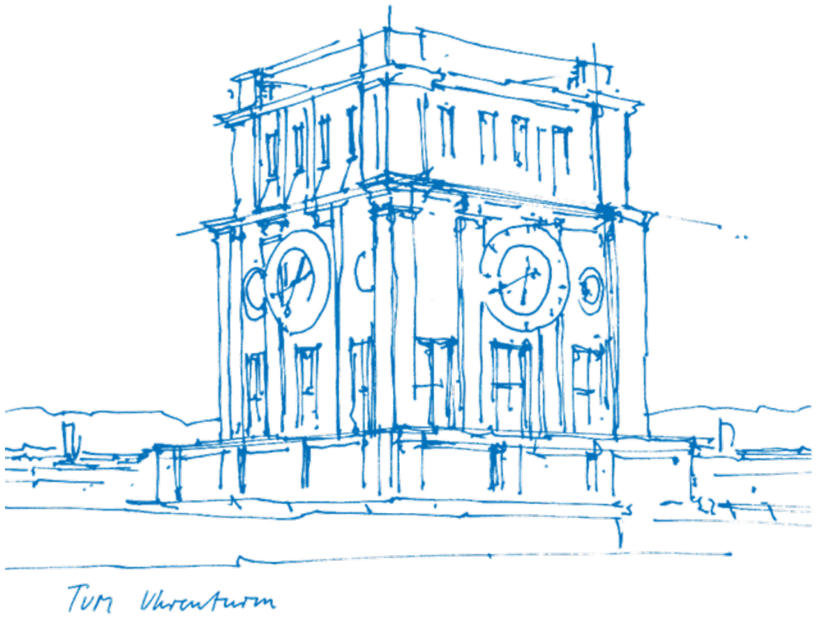
Chair of Network Architectures and Services
TUM Department of Informatics
Technical University of Munich (TUM)

GPLMT: A Lightweight Experimentation and Testbed Management Framework

Matthias Wachs, Nadine Herold, Stephan-A. Posselt, Florian Dold, Georg Carle

Heraklion, Greece, March 31, 2016



GPLMT

Not yet another testbed control tool . . .

Experiment automation is a valuable tool in research.

- Easy deployment
- Easy usage
- Shareable experiments
- Platform independent

GPLMT

Not yet another testbed control tool . . .

Experiment automation is a valuable tool in research.

Easy deployment	3 simple steps
Easy usage	Experiment description language
Shareable experiments	Encapsulation
Platform independent	SSH connections

GPLMT as a software solution supporting **experimentation control flow.**

Getting started with GPLMT

Basic experiment structure

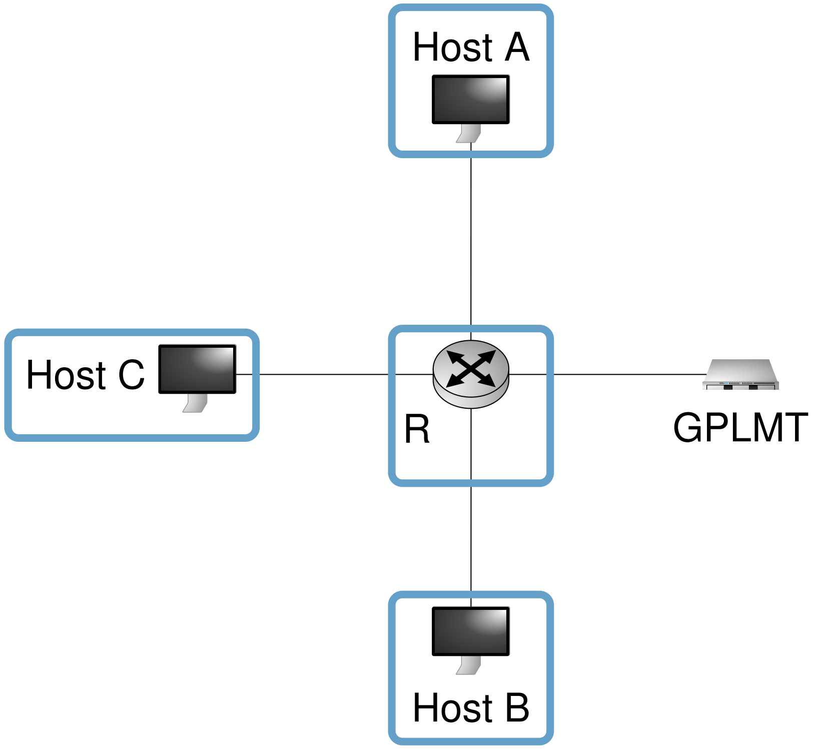
You have to define:

- **Who**
- is doing **what**
- and **when**

```
<experiment>
  <targets>
  </targets>

  <tasklists>
  </tasklists>

  <steps>
  </steps>
</experiment>
```

Getting started with GPLMT

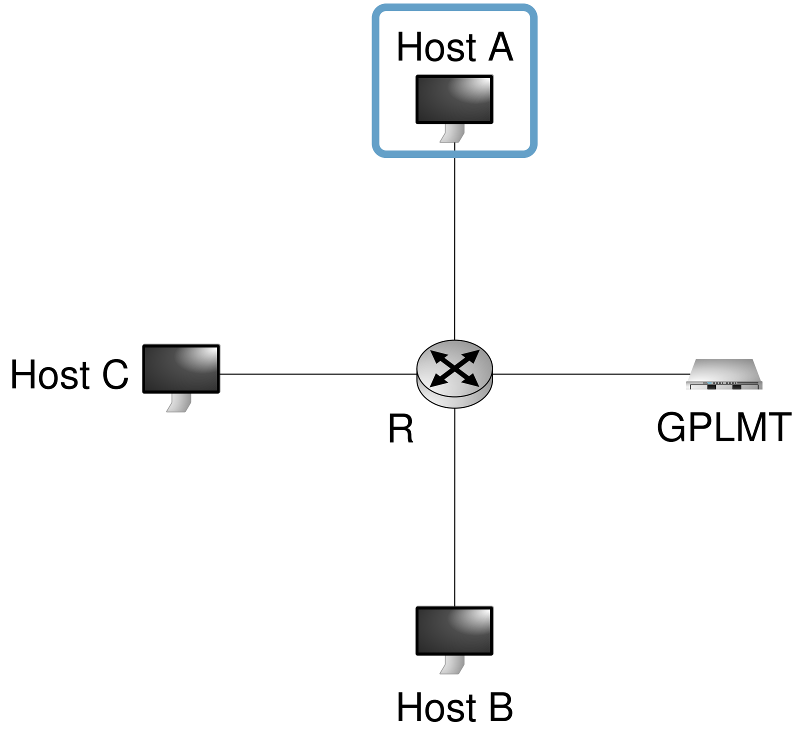
Defining targets



```
<experiment>
  <targets>


  </targets>

  <tasklists>
  </tasklists>

  <steps>
  </steps>
</experiment>
```

Getting started with GPLMT

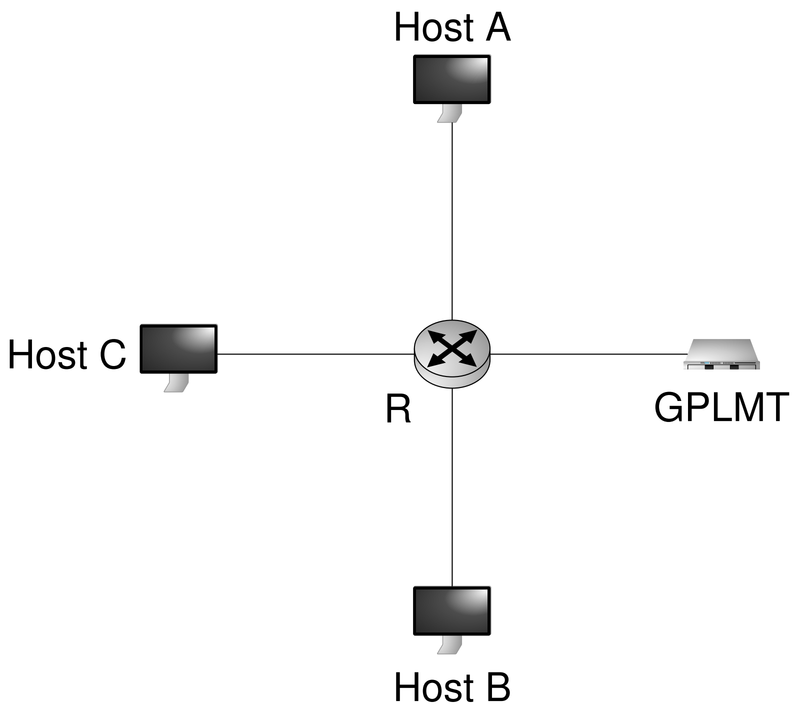
Defining targets



```
<experiment>
  <targets>
    <target name='A' type='ssh'>
      <user>testaccount</user>
      <host>A.example</host>
    </target>
  </targets>

  <tasklists>
  </tasklists>

  <steps>
  </steps>
</experiment>
```

Getting started with GPLMT

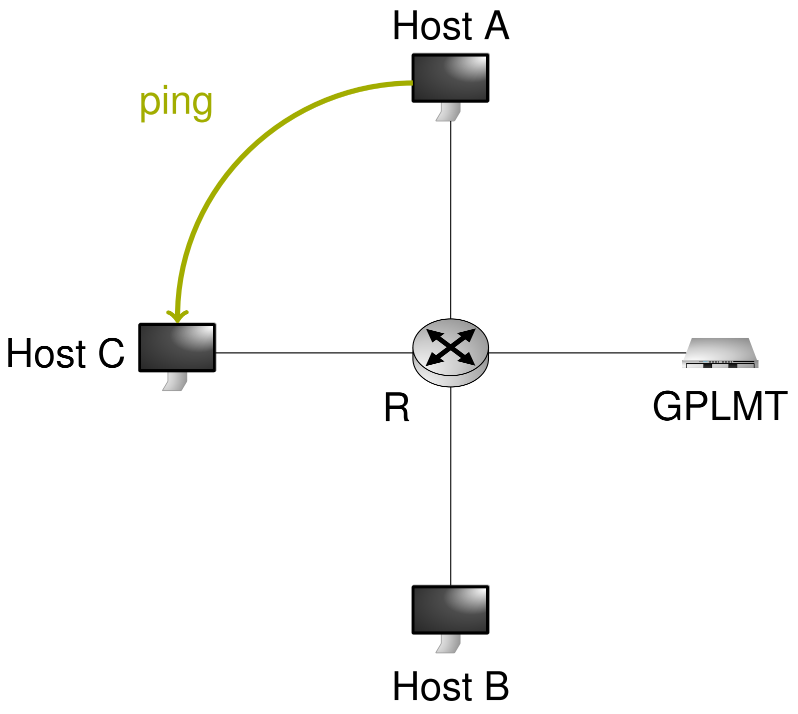
Defining tasks



```
<experiment>
  <targets>
  </targets>

  <tasklists>
    <tasklist name='doPing'>
      <seq><run>ping C.example -c 10
        ↪    </run></seq>
    </tasklist>
  </tasklists>

  <steps>
  </steps>
</experiment>
```

Getting started with GPLMT

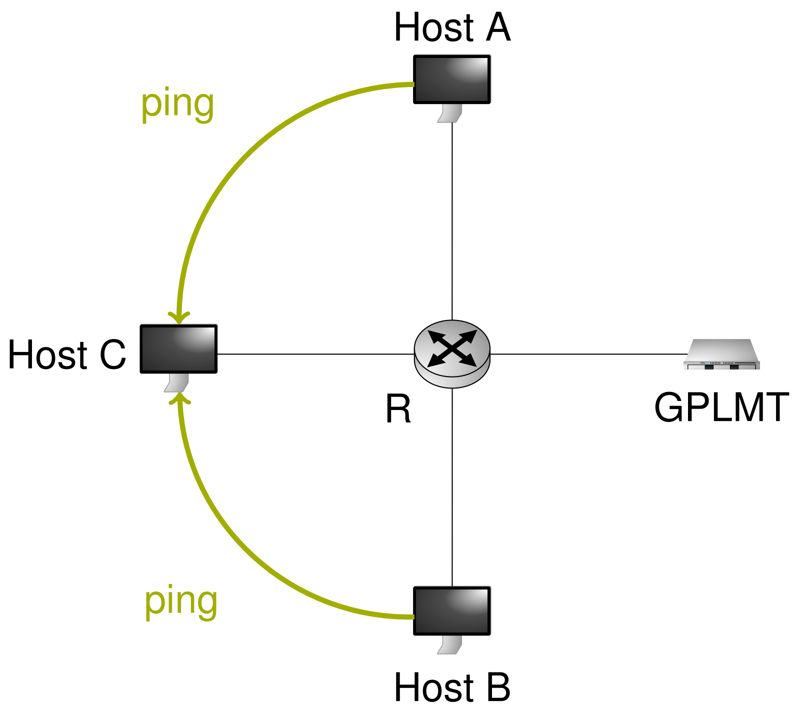
Putting things together



```
<experiment>
  <targets>
    <target name='A' type='ssh'>
  </targets>

  <tasklists>
    <tasklist name='doPing'>
  </tasklists>

  <steps>
    <step tasklist='doPing'
      ↪ targets='A' />
  </steps>
</experiment>
```

Doing some enhancements

A and B are going to ping C at the same time . . .

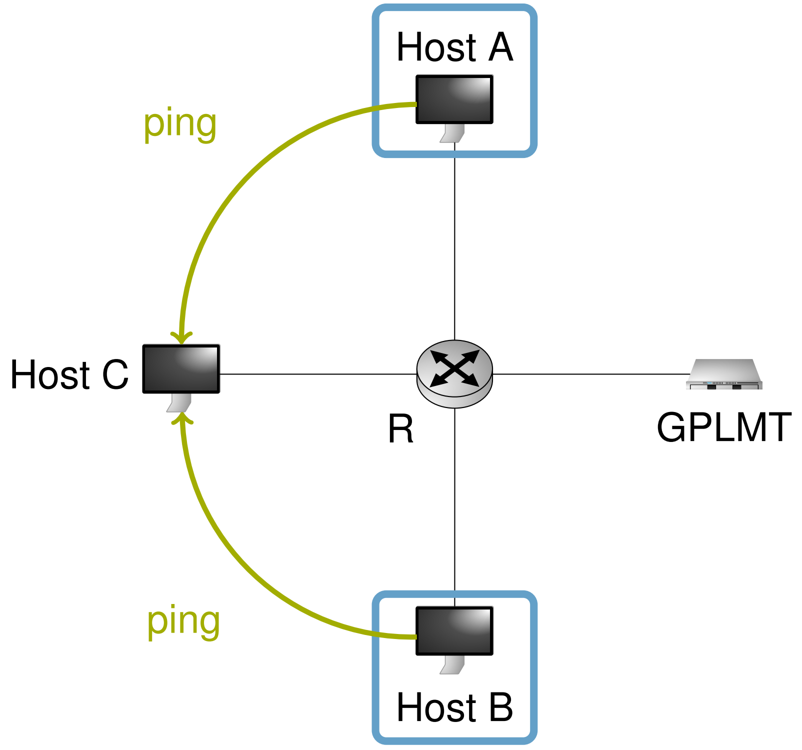


```
<experiment>
  <targets>
    ...
  </targets>

  <taskslists>
    ...
  </tasklists>

  <steps>
    <step tasklist='doPing'
      ↪ targets='A B' />
  </steps>
</experiment>
```

- Possible if no other actions are executed together and no more ping processes are around

Doing some enhancements

A and B are going to ping C at the same time . . .

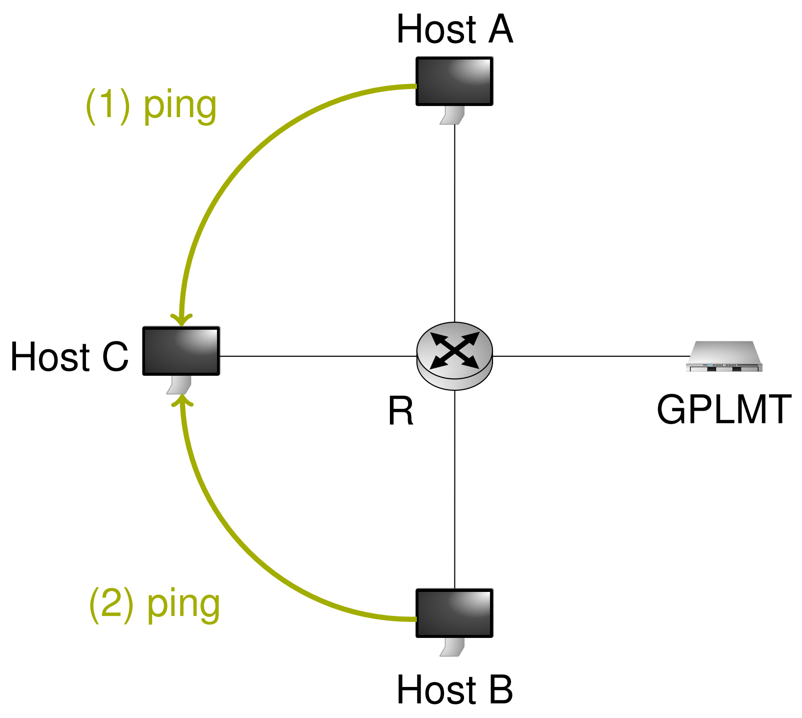


```
<experiment>
  <targets>
    <target name='G' type='group'>
      <target ref='A'/>
      <target ref='B'/>
    </target>
  </targets>

  <taskslists></tasklists>

  <steps>
    <step ... targets='G' />
  </steps>
</experiment>
```

- Better: extend the targets definition with a **group**.

Doing some enhancements

. . . or one after another in sequence

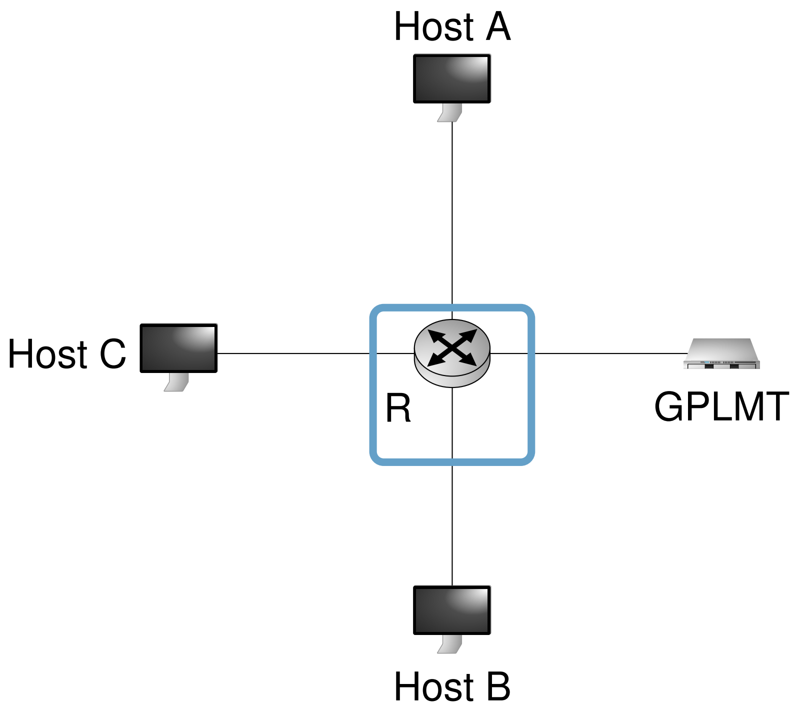


```
<experiment>
  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <step ... targets='A' />
    <synchonize />
    <step ... targets='B' />
  </steps>

</experiment>
```

- **synchronize** 'waits' until all steps are finished.

Having more fun with control flow

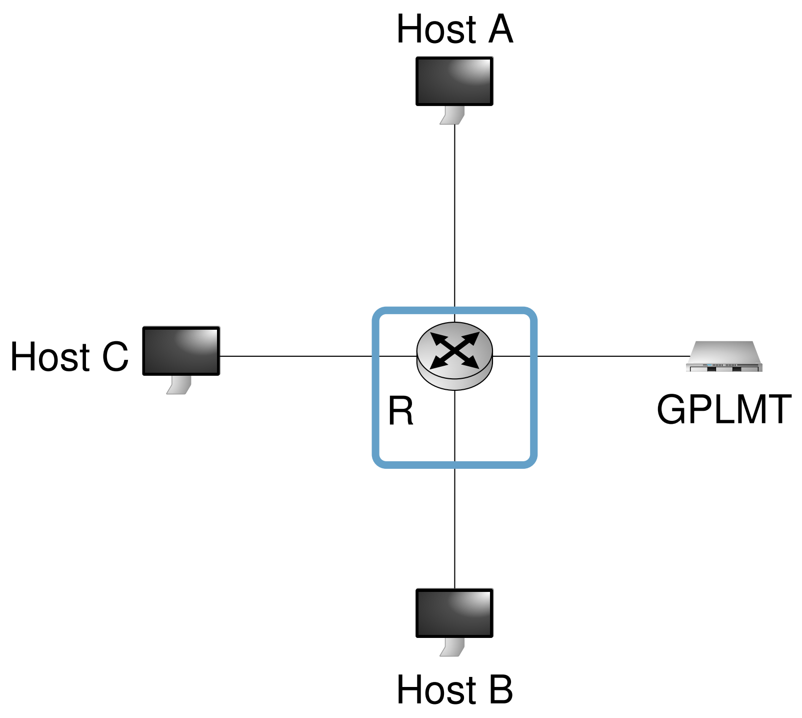
Monitor the experiment



```
<experiment>
  <targets>
  </targets>

  <taskslists>
    <tasklist name='monitor'>
      <seq><run>tcpdump -i any
          ↪ -w test.pcap</run></seq>
    </tasklist>
  </tasklists>

  <steps>
    <step tasklist='monitor'
       ↪ targets='R' />
  </steps>
</experiment>
```

Having more fun with control flow

How to collect the results? - Do cleanups!

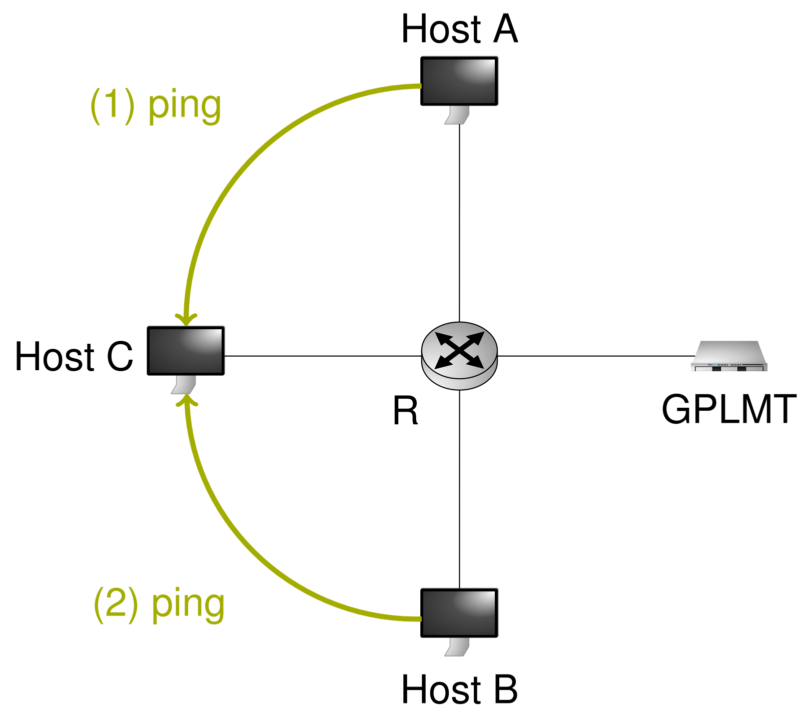


```
<experiment>

  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <register—teardown
      ↪ tasklist='x' targets='R'/>
    <step tasklist='monitor' .../>
  </steps>

</experiment>
```

- All **Teardowns** are registered and executed definitely at the end.

Having more fun with control flow

*Remember the **sychronize**?*

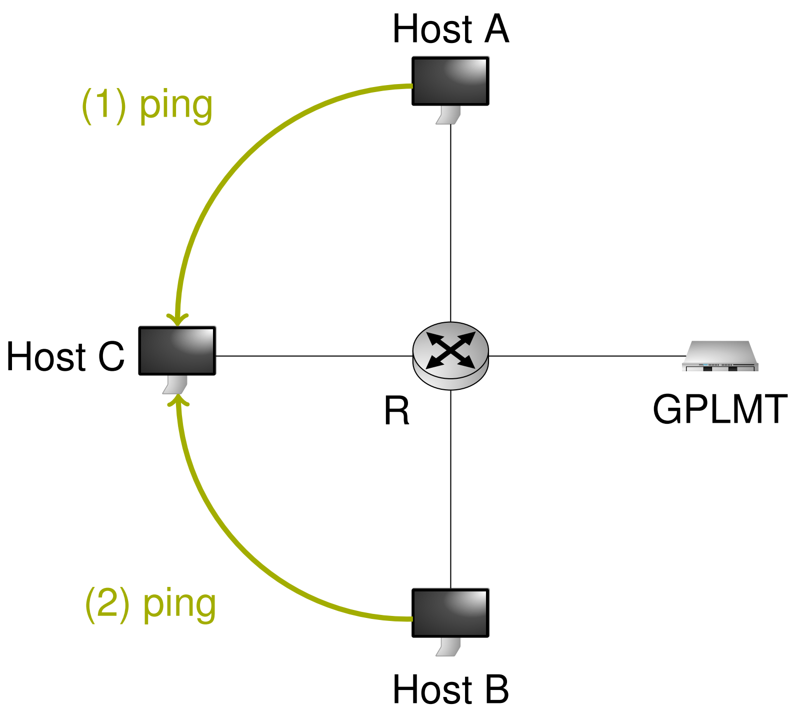


```
<experiment>
  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <register-teardown ... />
    <step tasklist='monitor' .../>
    <step ... targets='A' />
    <synchronize targets='A' />
    <step ... targets='B' />
  </steps>
</experiment>
```

- Possible: Define Targets to wait for.

Having more fun with control flow

*Remember the **sychronize**?*

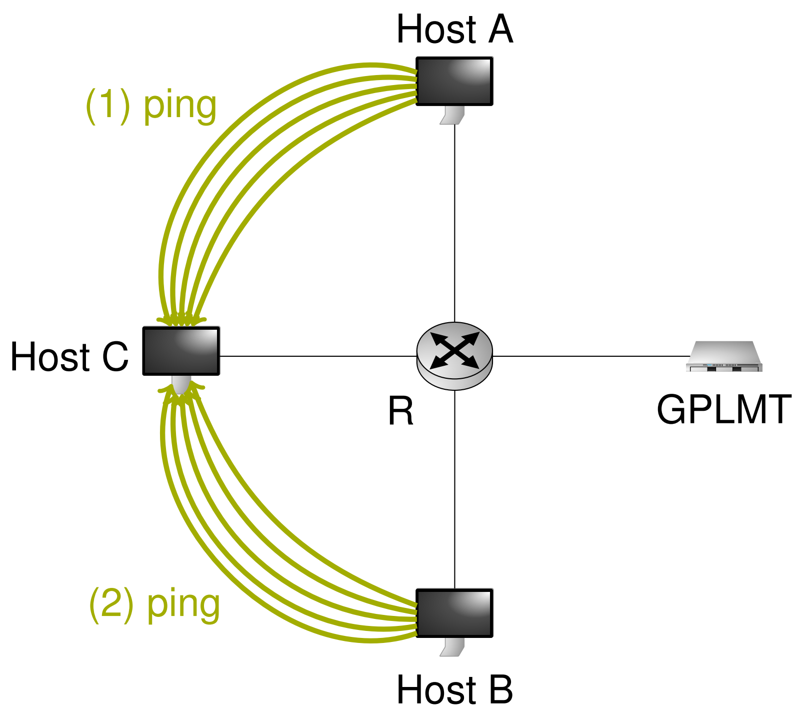


```
<experiment>

  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <register—teardown  ...  />
    <step  tasklist='monitor'
      ↪  background='true'
      ↪  targets='R'/>
  </steps>
</experiment>
```

- Better: Define a **Background Process** excluded from the Control Flow.

Having more fun with control flow

Multiple execution of tasks - Do not copy&paste

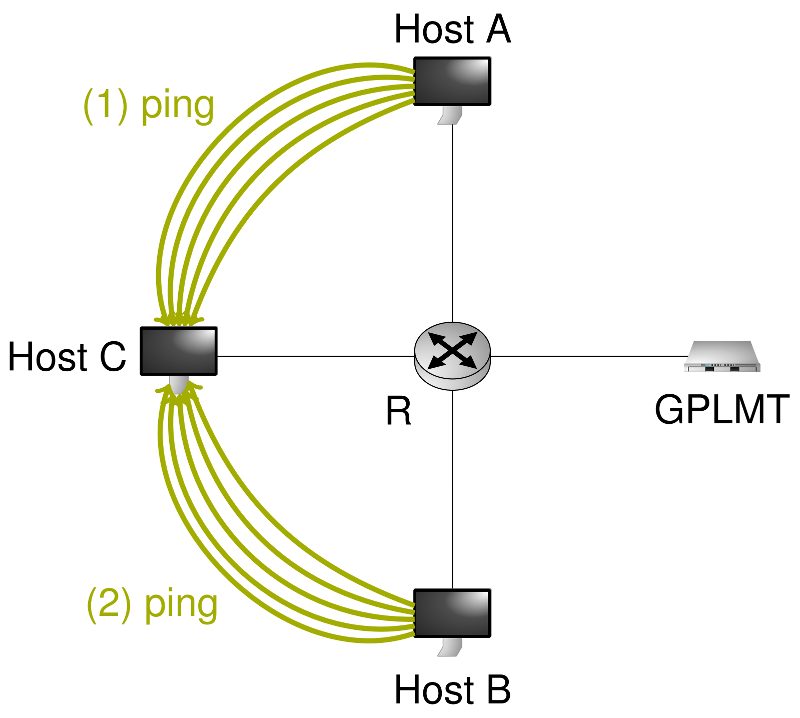


```
<experiment>
  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <step ... targets='A' />
    <synchronize />
    <step ... targets='B' />
    <synchronize />
    <step ... targets='A' />
    ...
  </steps>
</experiment>
```

- Hopefully, everybody is too lazy to do this.

Having more fun with control flow

Multiple execution of tasks - Do not copy&paste

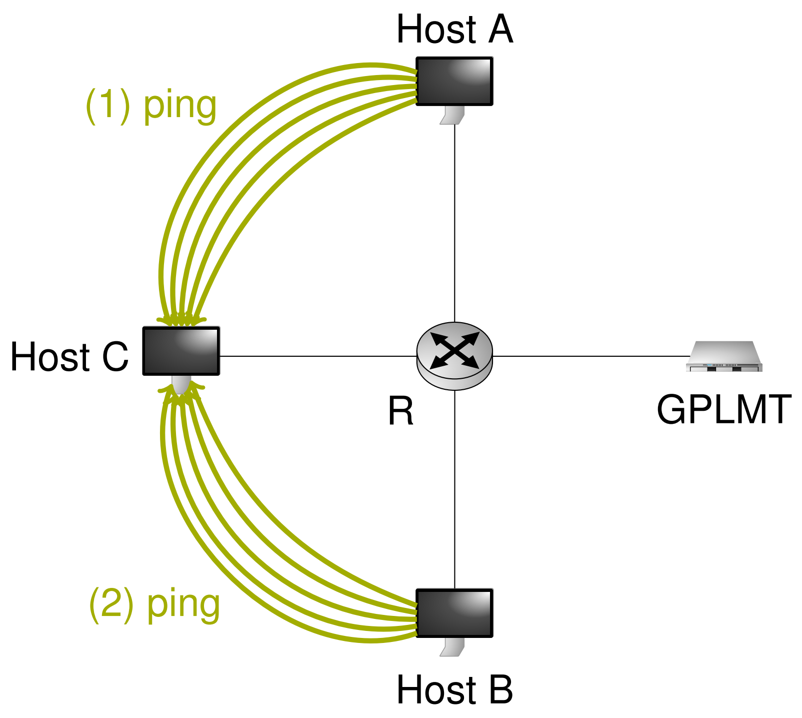


```
<experiment>
  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <loop repeat='5'>
      <step ... targets='A' />

      <step ... targets='B' />

    </loop>
  </steps>
</experiment>
```

- **All** pings are executed in **parallel**!

Having more fun with control flow

Multiple execution of tasks - Do not copy&paste



```
<experiment>
  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <loop repeat='5'>
      <step ... targets='A' />
      <synchronize />
      <step ... targets='B' />
      <synchronize />
    </loop>
  </steps>
</experiment>
```

- The second **synchronize** is mandatory here because of the loop semantic.

For the “lazy” ones

*Use **includes** for better modularity and reusability*

```
gplmt
├─ targets
│  ├─ hosts.xml
│  ├─ router.xml
│  └─ planetLab.xml
├─ tasklists
│  ├─ scan.xml
│  ├─ monitoring.xml
│  └─ teardowns.xml
└─ experiments
   └─ myFirstExperiment.xml
```

```
<experiment>
  <include file='../targets/
    ↪ hosts.xml' prefix='h'/>
  <include file='../taskslists/
    ↪ router.xml' prefix='r'/>
  <include file='../taskslists/
    ↪ scan.xml' prefix='s'/>

  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
  </steps>
</experiment>
```

For the “lazy” ones

*Use **includes** for better modularity and reusability*

```
gplmt
├─ targets
│  ├─ hosts.xml
│  ├─ router.xml
│  └─ planetLab.xml
├─ tasklists
│  ├─ scan.xml
│  ├─ monitoring.xml
│  └─ teardowns.xml
└─ experiments
   └─ myFirstExperiment.xml
```

```
<experiment>
  <include ... prefix='h'/>
  <include ... prefix='r'/>
  <include ... prefix='s'/>
  <targets>
    <target name='G' type='group'>
      <target ref='h.A'/>
      <target ref='h.B'/>
    </target>
    <target name='all' type='group'>
      <target  ref='G'/>
      <target  ref='r.R'/>
    </target>
  </targets>
  <taskslists></tasklists>
  <steps></steps>
</experiment>
```

For the “lazy” ones

*Use **includes** for better modularity and reusability*

```
gplmt
├─ targets
│  ├─ hosts.xml
│  ├─ router.xml
│  └─ planetLab.xml
├─ tasklists
│  ├─ scan.xml
│  ├─ monitoring.xml
│  └─ teardowns.xml
└─ experiments
   └─ myFirstExperiment.xml
```

```
<experiment>
  <include ... prefix='h'/>
  <include ... prefix='r'/>
  <include ... prefix='s'/>

  <targets>
  </targets>

  <taskslists>
  </tasklists>

  <steps>
    <step tasklist='s.doPing'
      ↪ targets='h.A'/>
  </steps>
</experiment>
```

Additional GPLMT Features in a nutshell

A quick and incomplete survey

More on **targets**:

- Local and PlanetLab targets
- Parametrization via per-target environment variables

More on **tasklists**:

- Parallel or sequential execution for tasklists
- Error handling strategies for tasklists (stop step, tasklist or experiment)
- Cleanup tasklists are supported
- Timeouts for tasklists

More on **steps**:

- Time-based loops (*during* and *until*)
- Time-triggered step execution (absolute and relative)

Conclusion

Wrapping things up

GPLMT is:

- Ready to use
- Open source and free sofware
- Publicly available on GitHub

GPLMT and its experiment definition language provides:

- Easy deployment and usage
- Shareable and reusable experiment definitions
- Platform-independent solutions

Questions?

Nadine Herold <herold@net.in.tum.de>
Matthias Wachs <matthias.wachs@tum.de>

What's next? – Give GPLMT a try:

`https://github.com/docmalloc/gplmt/`

Feedback, experiences and improvements are welcome!

Acknowledgments: This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS0145, project SURF.