FLAME
Flexible Lightweight Active Measurement Environment

Marcos L. Kirszenblatt Thiago B. Cardozo
Artur Ziviani Antônio Tadeu A. Gomes

National Laboratory for Scientific Computing (LNCC)

{marcoslk,thiagoc,ziviani,atagomes}@lncc.br

http://www.lncc.br/~ziviani

Introduction

Typical research fronts in active measurements

- development or improvement of measurement tools
- deployment of large-scale platforms

Problems in developing/improving active measurement tools

- large development and testing time
- low code reuse
- absence of standardized storage of results

Goal

A platform for the rapid prototyping
of active measurement tools that:

- offers basic primitives for active measurement
  Active measurement: involves the sending of probe packets
- is extensible
- stores the results in a centralized repository
- allows carrying out cooperative and uncooperative experiments
  Cooperative experiment: the tool must be executed at both ends
  Uncooperative experiment: the tool is executed only at the origin

Tools for active network measurement

- each tool targets a limited set of metrics
- low code reuse
- results are presented in an "ad hoc" way

Tools	COP	OWD	OWV	RTT	PLR	PRO	ROT	PHC	ECP	ABW
Iperf	✓		✓		✓					
owping	✓	✓			✓	✓				
pathchar			✓					✓		
pathload	✓									✓
pathrate	✓								✓	
ping				✓	✓					
QoSMet	✓	✓	✓		✓	✓				
sprobe									✓	
sting					✓	✓				
traceroute							✓			
traceroute-paris							✓			

Legend:
COP: tools that depend on cooperative destination nodes.
OWD: one-way delay; **OWV**: OWD variation; **RTT**: round-trip time;
PLR: packet loss rate; **PRO**: packet reordering; **ROT**: route tracing;
PHC: per-hop capactity; **ECP**: end-to-end link capacity;
ABW: available bandwidth.

Platforms for active network measurement

Manager-agent architecture

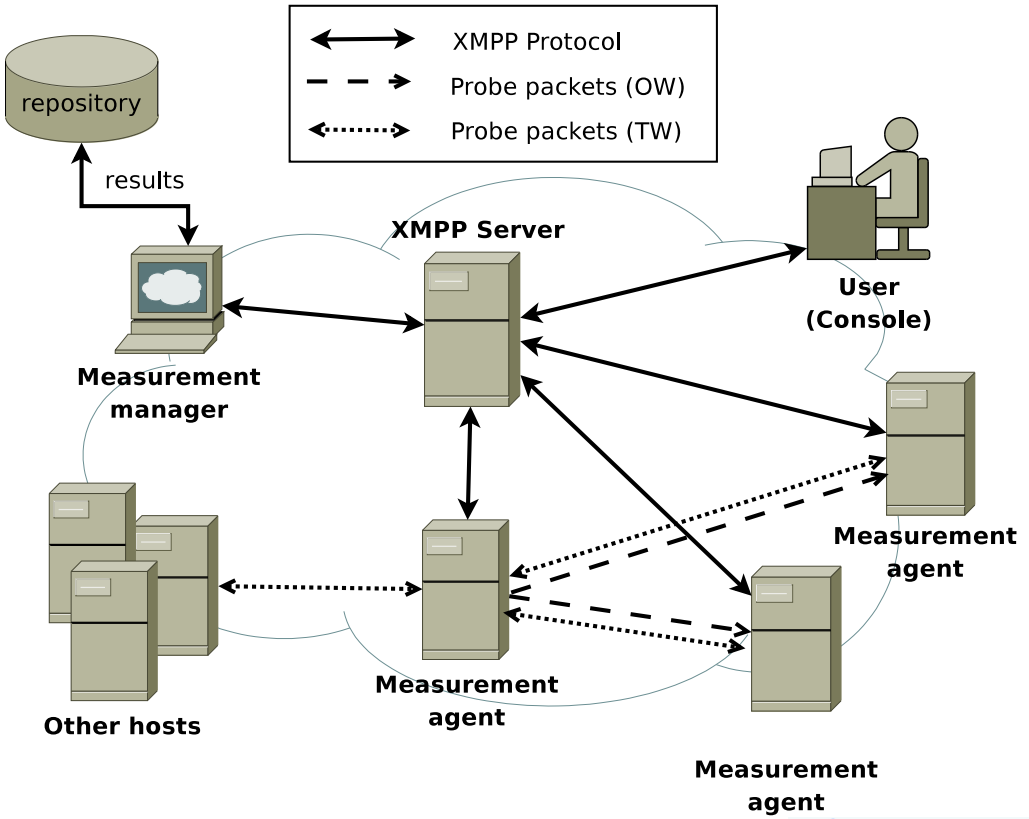
- NIMI
- ETOMIC
- DIMES
- NetQuest

- Scriptroute is the closest to ours
  - Ruby scripts
  - no centralized repository
  - no cooperative experiments

FLAME architecture

Components

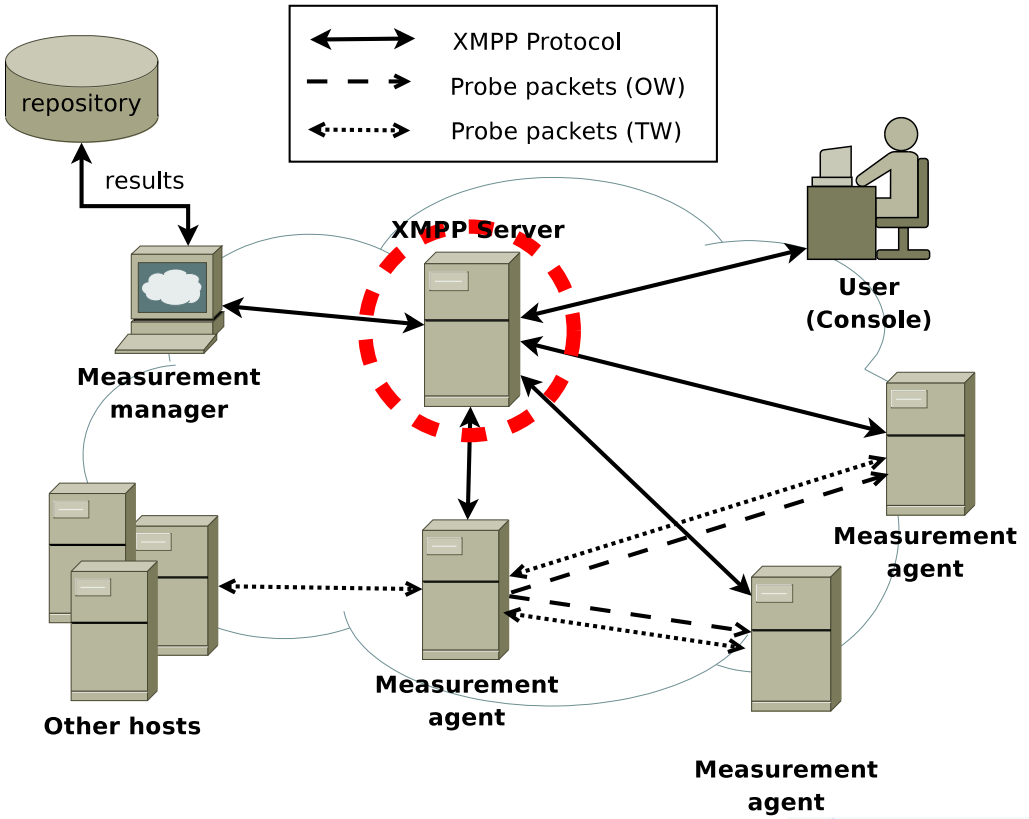
- XMPP server
- measurement manager
- user console
- measurement agents



FLAME architecture

XMPP server

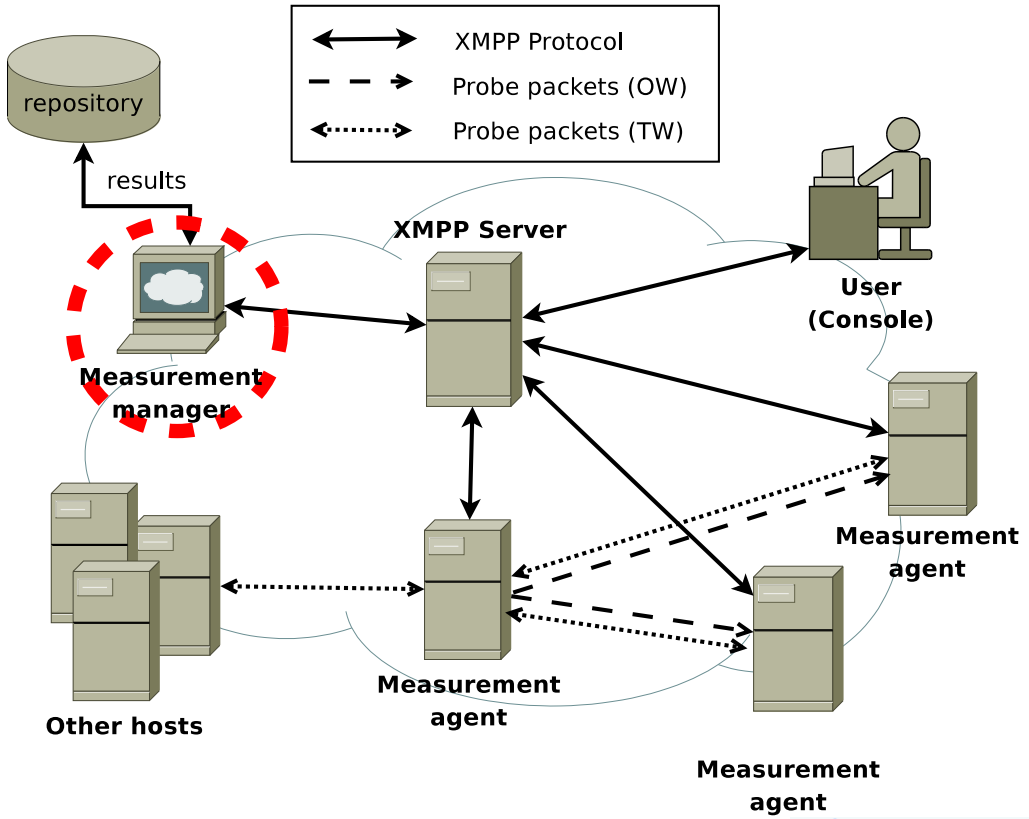
- acts as a message hub among other FLAME components
- presence control
- connectivity control
- offline message store for temporally disconnected components
- NAT/firewall friendly



FLAME architecture

Measurement manager

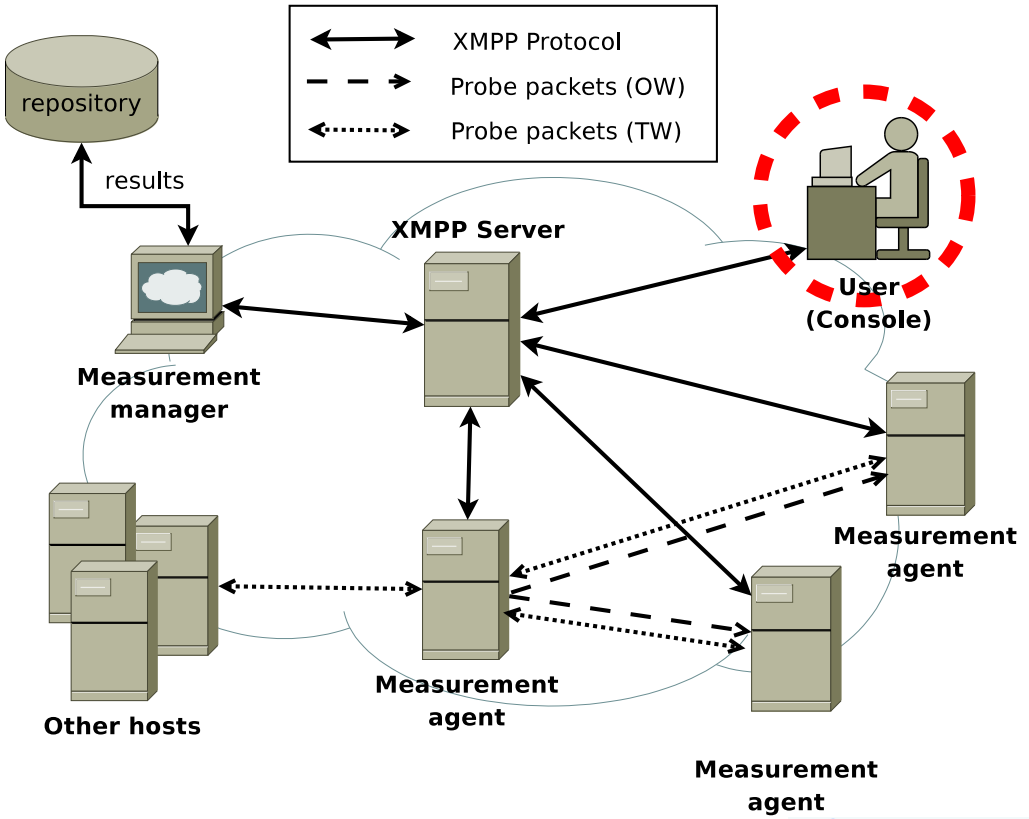
- controls the starting and the finishing of experiments
- only component that accesses the data repository



FLAME architecture

User console

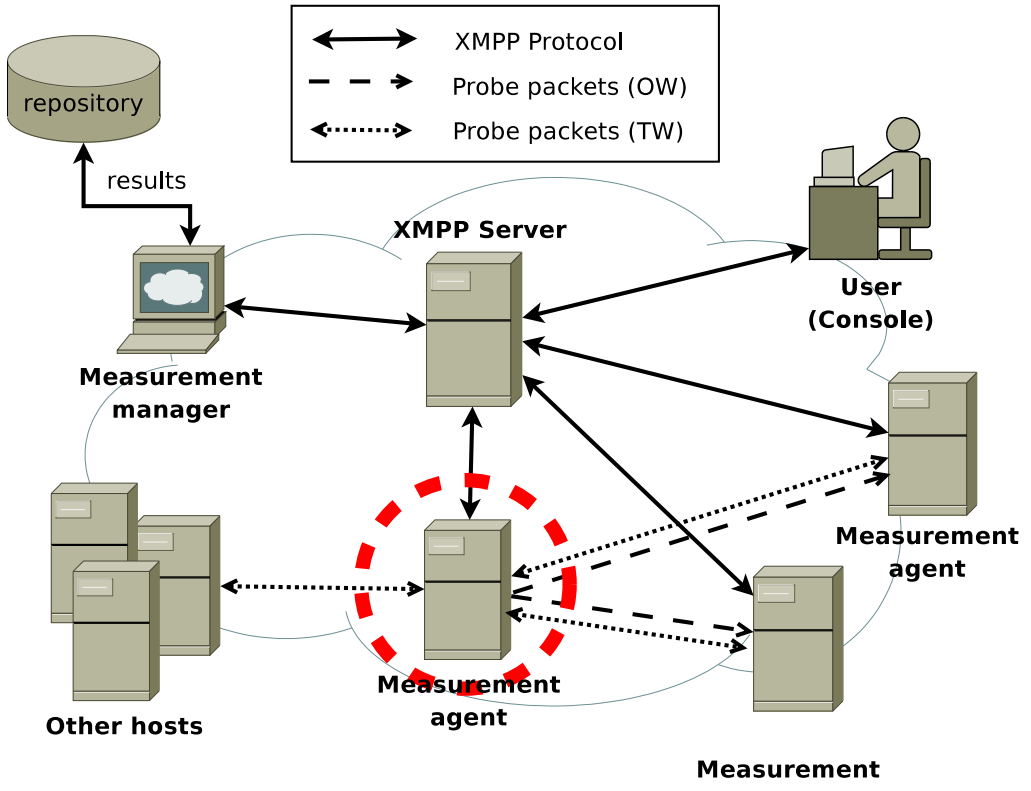
- registers an experiment at the manager
- allows access to specific agents
- sends commands or Lua scripts to the selected agent(s)



FLAME architecture

Measurement agents

- host a Lua interpreter
- run the scripts in a Sandbox environment
- provide access to a measurement API (LAMP)
- send the results to the manager
- may operate in exclusive mode



Why Lua?

Lua characteristics

- interpreted programming language
- simple syntax
- extensible semantics
- portability
- low memory footprint

Don't know Lua? Try it: `http://www.lua.org`

"Lua is a powerful, fast, lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping."

LAMP API: Lua Active Measurements Primitives

The active measurement primitives are implemented in C, being available through a minimalist API

LAMP API

lamp.sendUDPOW ( )
lamp.sendUDPTW ( )
lamp.sendTCPOW ( )
lamp.sendTCPTW ( )
lamp.sendICMPTW ( )
lamp.sendUDPPT ( )

Notation

OW = One Way (Cooperative)
TW = Two Way (Uncooperative)
PT = Packet Train (Cooperative)

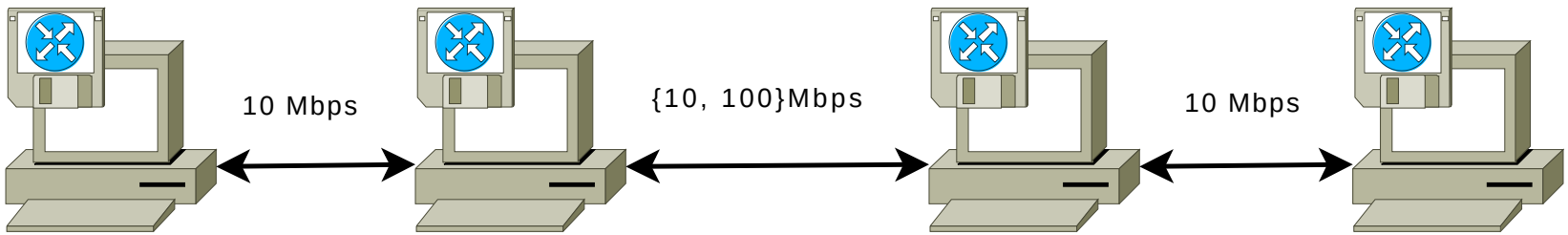
Experimental scenarios

Source and target nodes of the Planet-lab experiments

	IP address	Domain name
Source		
S1	200.19.159.34	`planetlab1.pop-mg.rnp.br`
Targets		
T1	130.83.166.198	`host1.planetlab.informatik.tu-darmstadt.de`
T2	128.112.139.80	`alice.cs.princeton.edu`
T3	132.65.240.100	`planet1.cs.huji.ac.il`
T4	130.216.1.22	`planetlab-1.cs.auckland.ac.nz`
T5	131.112.243.102	`node2.planet-lab.titech.ac.jp`

Experimental scenarios

Local testbed



Mac Minis running Quagga over Ubuntu 9.04

FLAME ping prototype

```
1  function flamePing(params)
2    -- Ping params (and corresponding defaults)
3    local _target = params.target or "127.0.0.1"
4    local _size = params.size or 56
5    local _interval = params.interval or 250000 -- microsecond resolution
6    local _npackets = params.npackets or 10
7    local _protocol = params.protocol or "icmp"
8    local _timeout = params.timeout or 5000000 -- microsecond resolution
9    local _ttl = params.ttl or 30
10
11   for i = 1,_npackets do
12     local _response
13
14     -- Choose probing protocol (for UDP and TCP primitives,
15     --  since port is not indicated it is randomly chosen above 1024)
16     if _protocol == "icmp" then
17       _response=lamp.sendICMPTW{ip=_target, size=_size,
18                                 timeout=_timeout, ttl=_ttl}
19     elseif _protocol == "udp" then
20       _response=lamp.sendUDPTW{ip=_target, size=_size,
21                                timeout=_timeout, ttl=_ttl}
22     elseif _protocol == "tcp" then
23       _response=lamp.sendTCPTW{ip=_target, size=_size,
24                                timeout=_timeout, ttl=_ttl}
25     else
26       print("INVALID PROTOCOL:", _protocol)
27       return
28     end
29
30     -- Check host/net unreachability
31     if _response and _response.loss == ICMP_HOST_UNREACH then
32       print("DESTINATION UNREACHABLE: ", _target)
33       return
34     end
35
36     -- Wait time between probes
37     if type(_interval) == "number" then
38       lamp.sleep(_interval)
39     elseif type(_interval) == "table" then
40       lamp.sleep(_interval.func(_interval.params))
41     end
42   end --for i
43 end
```

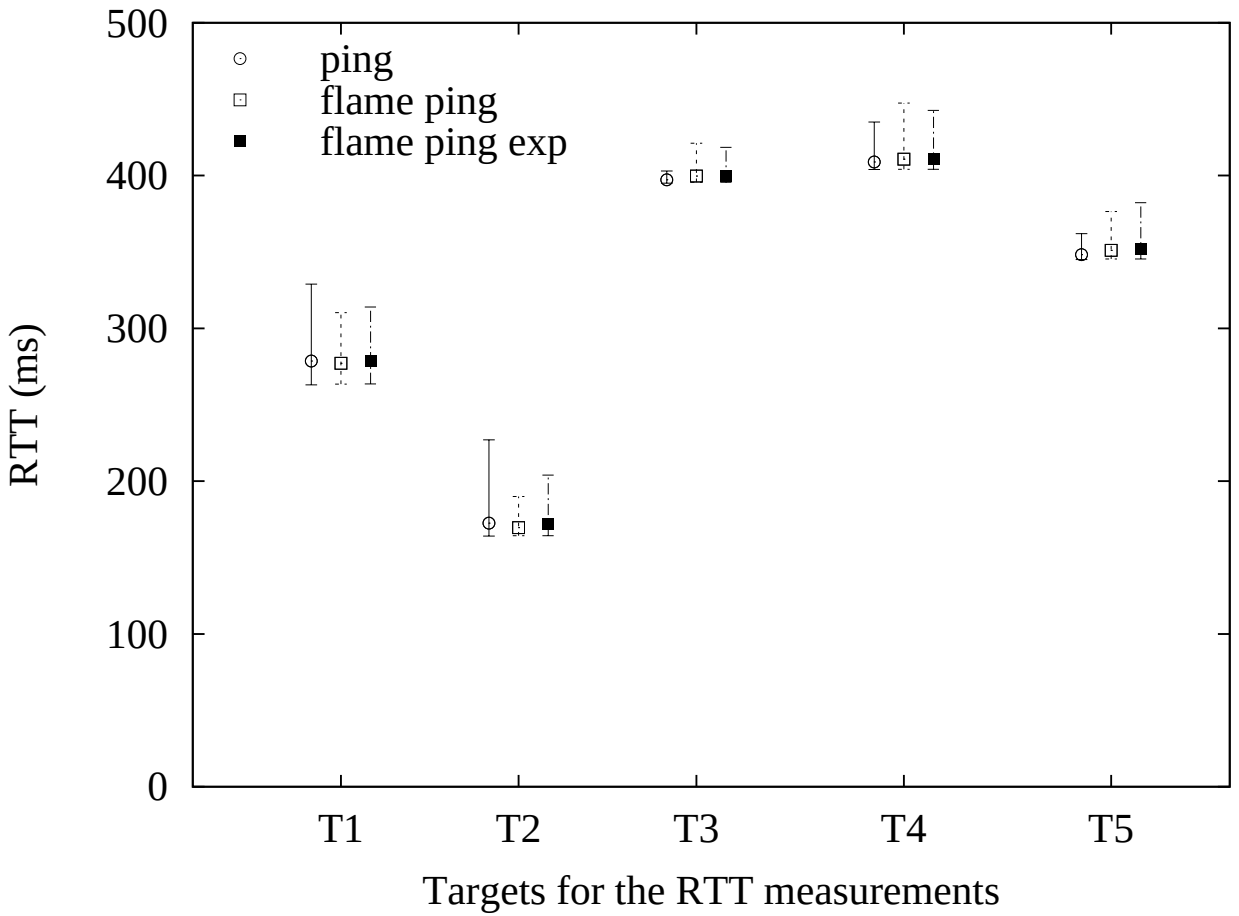
Commands to execute `flamePing` over Planet-lab nodes

Probes uniformly spaced	Probes exponentially spaced

```
local targets = {
  "130.83.166.198",
  "128.112.139.80",
  "132.65.240.100",
  "130.216.1.22",
  "131.112.243.102"
}

for k,v in ipairs(targets) do
  flamePing{target=v}
end
```

```
local function expdist(lambda)
  local r = 0
  repeat r = math.random() until(r ~= 0)
  return math.floor(-math.log(r)/lambda)
end

local targets = ... -- same as left

for k,v in ipairs(targets) do
  flamePing{
    target=v,
    interval={ func=expdist, params=1/10000000 }
  }
end
```

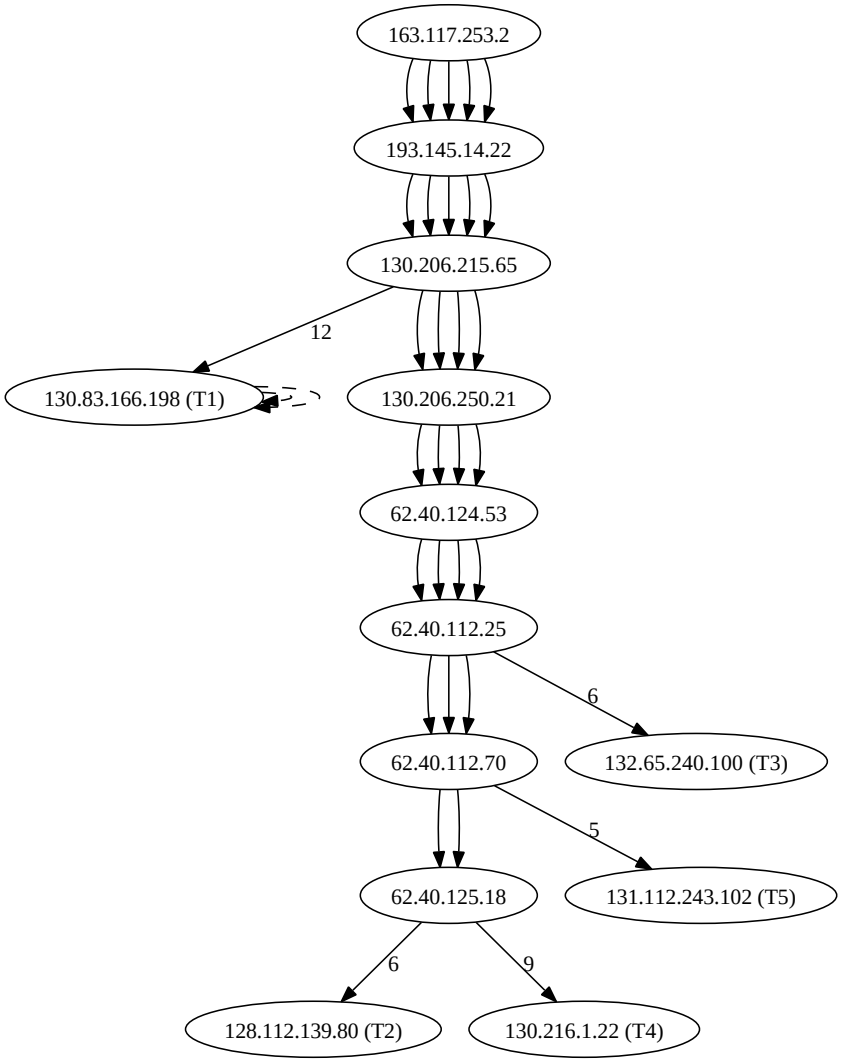
RTT measurement statistics in Planet-lab



FLAME tracing route prototype

```
1  function flameTrace(params)
2    local _target = params.target or "127.0.0.1"
3    local _size = params.size or 60
4    local _interval = params.interval or 250000
5    local _npackets = params.npackets or 3
6    local _protocol = params.protocol or "udp"
7    local _timeout = params.timeout or 5
8    local _maxhops = params.maxhops or 30
9
10   for h = 1, _maxhops do
11     local _response
12
13     for i = 1,_npackets do
14       -- Choose probing protocol (for UDP and TCP primitives,
15       --  since port is not indicated it is randomly chosen above 1024)
16       if _protocol == "icmp" then
17         _response=lamp.sendICMPTW{ip=_target, size=_size,
18                                   timeout=_timeout, ttl=h}
19       elseif _protocol == "udp" then
20         _response=lamp.sendUDPTW{ip=_target, size=_size,
21                                  timeout=_timeout, ttl=h}
22       elseif _protocol == "tcp" then
23         _response=lamp.sendTCPTW{ip=_target, size=_size,
24                                  timeout=_timeout, ttl=h}
25       else
26         print("INVALID PROTOCOL:", _protocol)
27         return
28       end
29
30       ... -- same as lines 30-41 in flamePing
31     end --for i
32
33     -- Is current node the target?
34     if (_response.dstIP == _target) then break end
35   end --for h
36 end
```

Graph representation of `traceroute` and `flameTrace` results



The numbered edges indicate the number of omitted hops between the corresponding nodes

FLAME one-way delay measurement prototype

```
1  function flameOWD(params)
2    local _target = params.target or "127.0.0.1"
3    local _size = params.size or 56
4    local _interval = params.interval or 750000
5    local _npackets = params.npackets or 5
6    local _protocol = params.protocol or "udp"
7    local _timeout = params.timeout or 5000000
8    local _port = params.port or nil
9
10   for i = 1,_npackets do
11     local _response
12
13     -- Choose probing protocol (if port is not indicated the destination node
14     -- chooses a port above 1024. In any case, for OWD operations,
15     -- the source and destination nodes agree on the used port
16     -- through XMPP messages)
17     if _protocol == "udp" then
18       _response=lamp.sendUDPOWD{ip=_target, size=_size,
19                                 timeout=_timeout, port=_port}
20     elseif _protocol == "tcp" then
21         _response=lamp.sendTCPOWD{ip=_target, size=_size,
22                                 timeout=_timeout, port=_port}
23     else
24       print("INVALID PROTOCOL:", _protocol)
25       return
26     end
27
28     -- Check port unavailability and host/net unreachability
29     if _response then
30       if _response.loss == ICMP_HOST_UNREACH then
31         print("DESTINATION UNREACHABLE: ", _target)
32         return
33       elseif _response.loss == PORT_ALREADY_IN_USE then
34         print("DESTINATION PORT ALREADY IN USE: ", _target, _port)
35         return
36       end
37     end
38
39     ... -- same as lines 36-41 in flamePing
40   end --for i
41 end
```

Measured one-way delay in ms

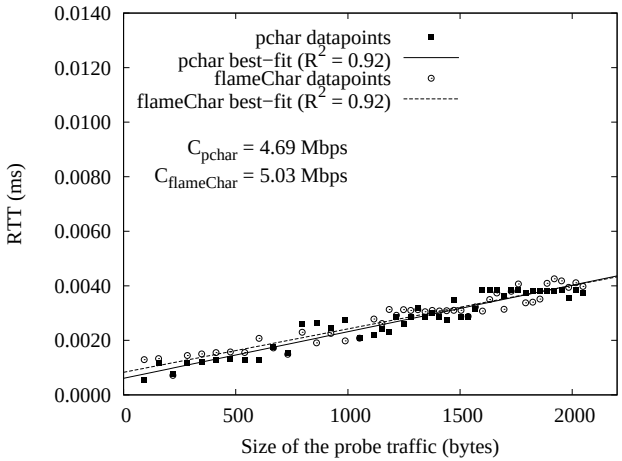
	10-10-10 Topology			10-100-10 Topology		
	P05	Median	P95	P05	Median	P95
`owping`	1.28	1.39	1.50	1.10	1.21	1.32
`flameOWD`	0.87	0.91	0.98	0.46	0.51	0.58

NTP estimated synch error is $\pm$ 0.41 ms

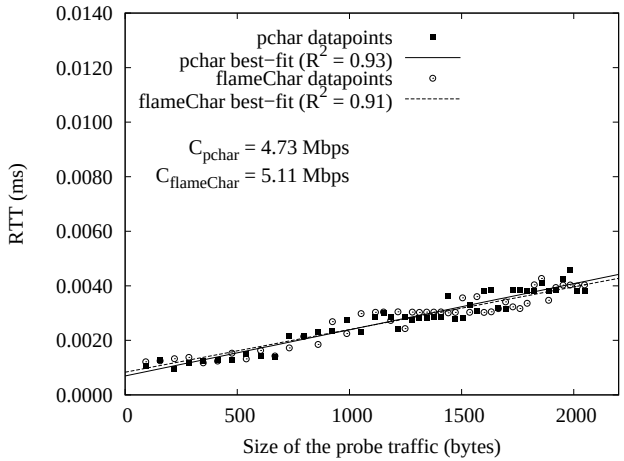
FLAME link capacity estimation prototype

```
1  function flameChar(params)
2    local _mtu = params.mtu or 1500 -- probe size limit
3    local _increment = params.increment or 32 -- difference between probe sizes
4    local _npackets = math.floor(_mtu/_increment)
5    local _maxhops = params.maxhops or 30 -- maximum number of allowed hops
6    local _repetitions = params.repetitions or 32 -- number of tests per hop
7    local _target = params.target or "127.0.0.1"
8    local _timeout = params.timeout or 3
9    local _interval = param_table.interval or 500000 -- time between probes
10   local _sizelist = generateProbeSizeList(_npackets, _increment)
11
12   for h = 1, _maxhops do
13     local _response
14     for t = 1,_repetitions do
15       for i = 1,_npackets do
16         local _response
17
18         -- Choose probing protocol (for UDP and TCP primitives,
19         --  since port is not indicated it is randomly chosen above 1024)
20         if _protocol == "icmp" then
21           _response = lamp.sendICMPTW{ip=_target, size=_sizelist[i], ttl=h, timeout=_timeout}
22         elseif _protocol == "udp" then
23           _response = lamp.sendUDPTW{ip=_target, size=_sizelist[i], ttl=h, timeout=_timeout}
24         elseif _protocol == "udp" then
25           _response = lamp.sendTCPTW{ip=_target, size=_sizelist[i], ttl=h, timeout=_timeout}
26         else
27           print("INVALID PROTOCOL:", _protocol)
28           return
29         end
30
31         ... -- same as lines 30-41 in flamePing
32       end --for i
33     end --for t
34
35     -- Is current node the target?
36     if (_response.remIP == _target) then break end
37   end --for h
38 end
39
40 -- Function to create a shuffled package size list
41 function generateProbeSizeList(npackets, increment)
42   ... -- implementation omitted due to space limitations
43 end
```

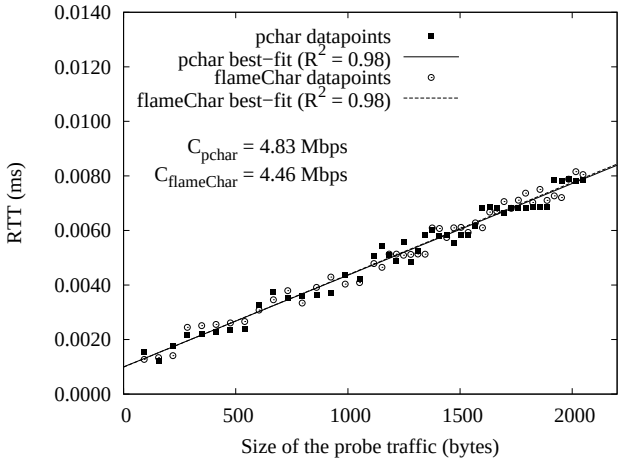
Link capacity measurements



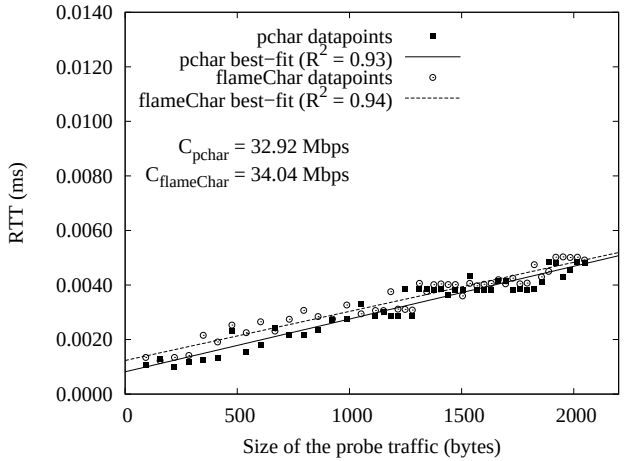
(a) Link 1, 10-10-10 Topology



(b) Link 1, 10-100-10 Topology



(c) Link 2, 10-10-10 Topology



(d) Link 2, 10-100-10 Topology

Conclusions

FLAME provides the following contributions

- an environment for the rapid prototyping of active measurement tools based on a small but comprehensive set of probing primitives, allowing the implementation of tools that depend or not on cooperative destination nodes;
- a centralized repository for implicitly gathering measurement results in a common data format, encouraging the untangling of probing and data output functionality and easing the process of further analysis and comparison among different result datasets.

Conclusions

Possible uses of FLAME prototyping

- besides rapid prototyping of new active measurement techniques...
- fair comparison among different measurement techniques
- easier combination of results from different tools
- easier tailoring of FLAME-based prototypes

Future work

It's an ongoing work!

Future work

1. dynamic selection of available agents
2. enrichment of the measurement toolkit with "usual" active measurement tools

Application in other contexts

- porting measurement agents to resource-constrained devices
  - taking benefit of Lua's low memory footprint

That's all, folks!

Interested? Give FLAME a try!

FLAME code is available at

`http://martin.lncc.br/main-software-flame`

Contact

Marcos Kirszenblatt: marcoslk@lncc.br

Thiago Cardozo: thiagoc@lncc.br

Artur Ziviani: ziviani@lncc.br

Antônio Tadeu Azevedo Gomes: atagomes@lncc.br