An Integrated Active Measurement Programming Environment

A design for a next-generation Internet Measurement Infrastructure

Matthew Luckie

Brendon Jones

Shivani Hariprasad

Ken Keys

k claffy

Raffaele Sommese

Ricky Mok

PAM 2025

caida

Outline

- Problem: unmet need for a sophisticated, wide-scale, topologically diverse active measurement infrastructure.
- Spectrum of possible solutions; we'll review these.

Outline

- Problem: unmet need for a sophisticated, wide-scale, topologically diverse active measurement infrastructure.
- Spectrum of possible solutions; we'll review these.
- Domain Specific Language (DSL) discussion at AIMS May 2023 workshop occurred over multiple days. Idea:
  - expose a set of measurement primitives (e.g., DNS, ping, traceroute, HTTP)
  - expose a set of measurement vantage points
  - users write applications that use these primitives enhanced with their logic
- In this talk, we'll outline our approach, architecture, implementation, and suggested code structures. We invite researchers to use our platform.

Solution Spectrum

Least Restrictive

Shell access to VPs	*PlanetLab*
Run code in containers on the VPs	*EdgeNet*
Run code to construct packet sequences in sandbox on the VP	*Scriptroute*
VPN access to send packets from VPs, logic off-VP	*PacketLab*
An integrated active measurement programming environment, logic on-VP, or in infrastructure / Domain Specific Language (DSL)	
API to use measurement primitives, logic elsewhere	*Atlas, Ark*
Use provided data	*Atlas, Ark*

Most Restrictive

Our Platform Goals

1. **Easy to use.** The protocol stack, implementation, programming, and sysadmin expertise required for measurement should be minimized.

Our Platform Goals

1. **Easy to use.** The protocol stack, implementation, programming, and sysadmin expertise required for measurement should be minimized.
2. **Performant.** The delay between measurement and result should be small, so that researchers can build complex reactive measurements.

Our Platform Goals

1. **Easy to use.** The protocol stack, implementation, programming, and sysadmin expertise required for measurement should be minimized.
2. **Performant.** The delay between measurement and result should be small, so that researchers can build complex reactive measurements.
3. **Interoperable.** The components that we use should be off-the-shelf and easily deployable to maximize avenues of future deployment.

Our Platform Goals

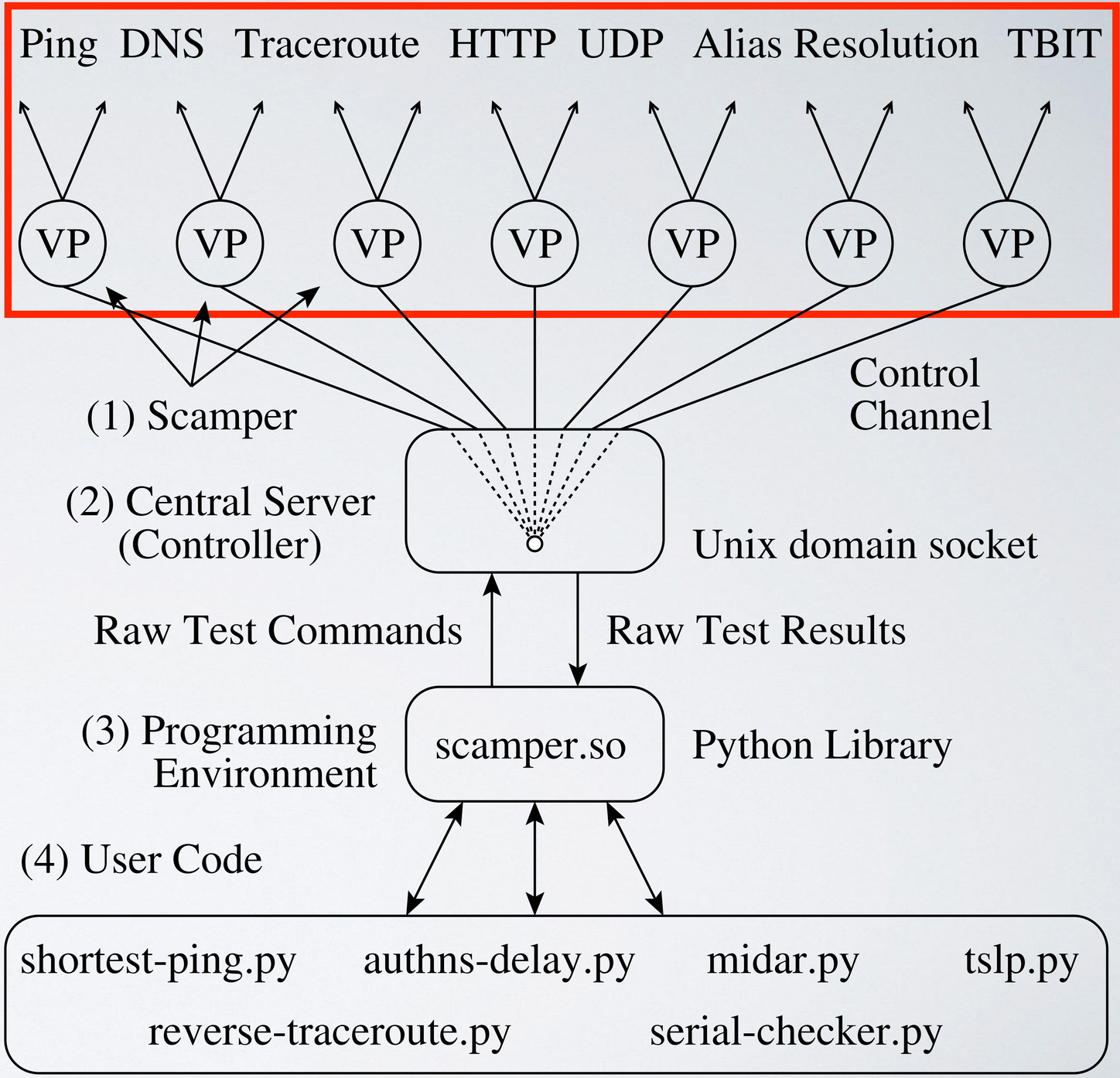
1. **Easy to use.** The protocol stack, implementation, programming, and sysadmin expertise required for measurement should be minimized.
2. **Performant.** The delay between measurement and result should be small, so that researchers can build complex reactive measurements.
3. **Interoperable.** The components that we use should be off-the-shelf and easily deployable to maximize avenues of future deployment.
4. **Site-host transparent.** The environment should allow platform operators to accurately describe the types of measurements the VPs will do.

We deployed our platform onto CAIDA's Ark infrastructure, and made all components open source.

System Architecture

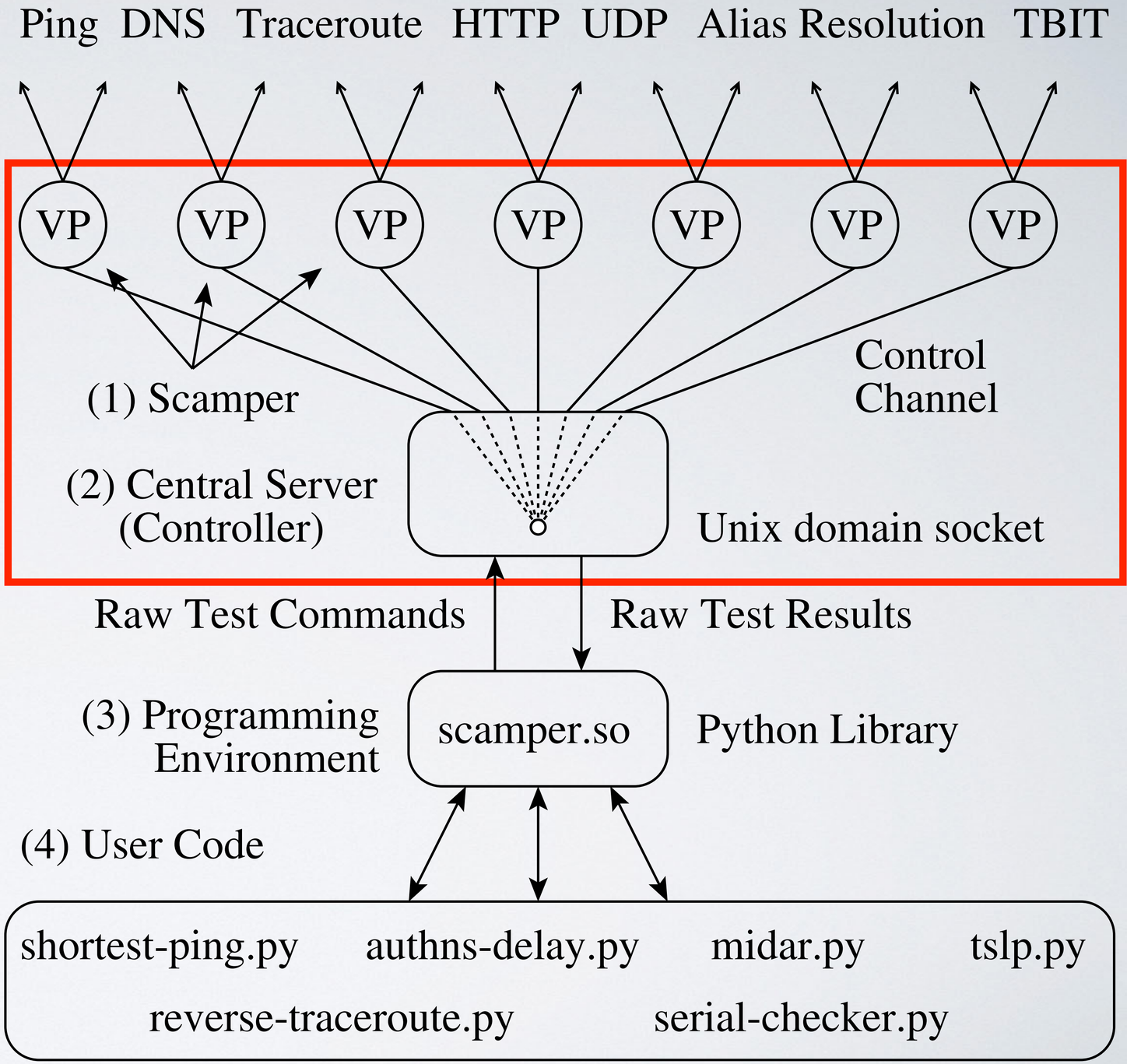
System Architecture

We used scamper to provide measurement primitives on the VPs.



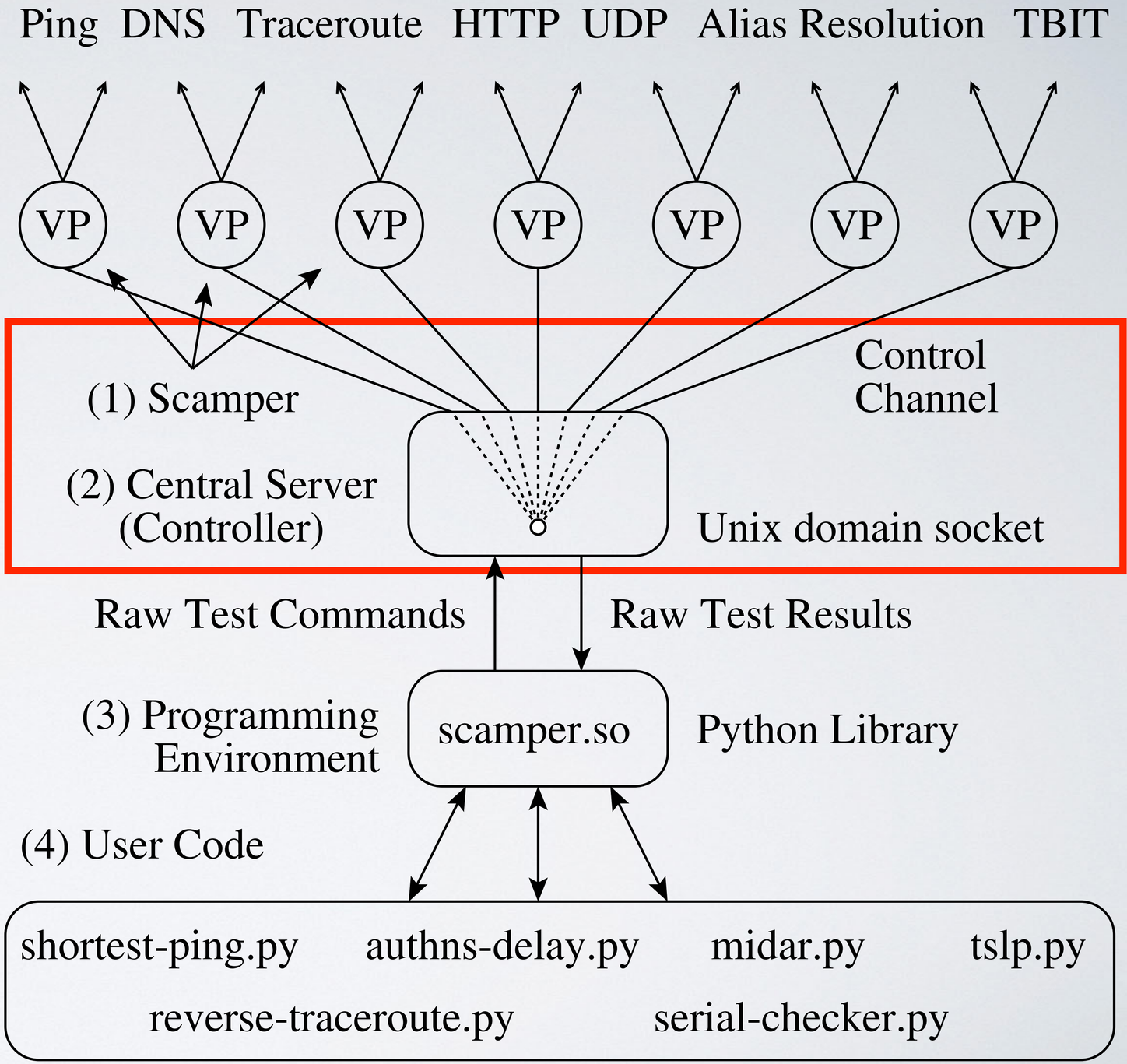
System Architecture

We configured scamper on each VP to connect back to a central server at CAIDA.



System Architecture

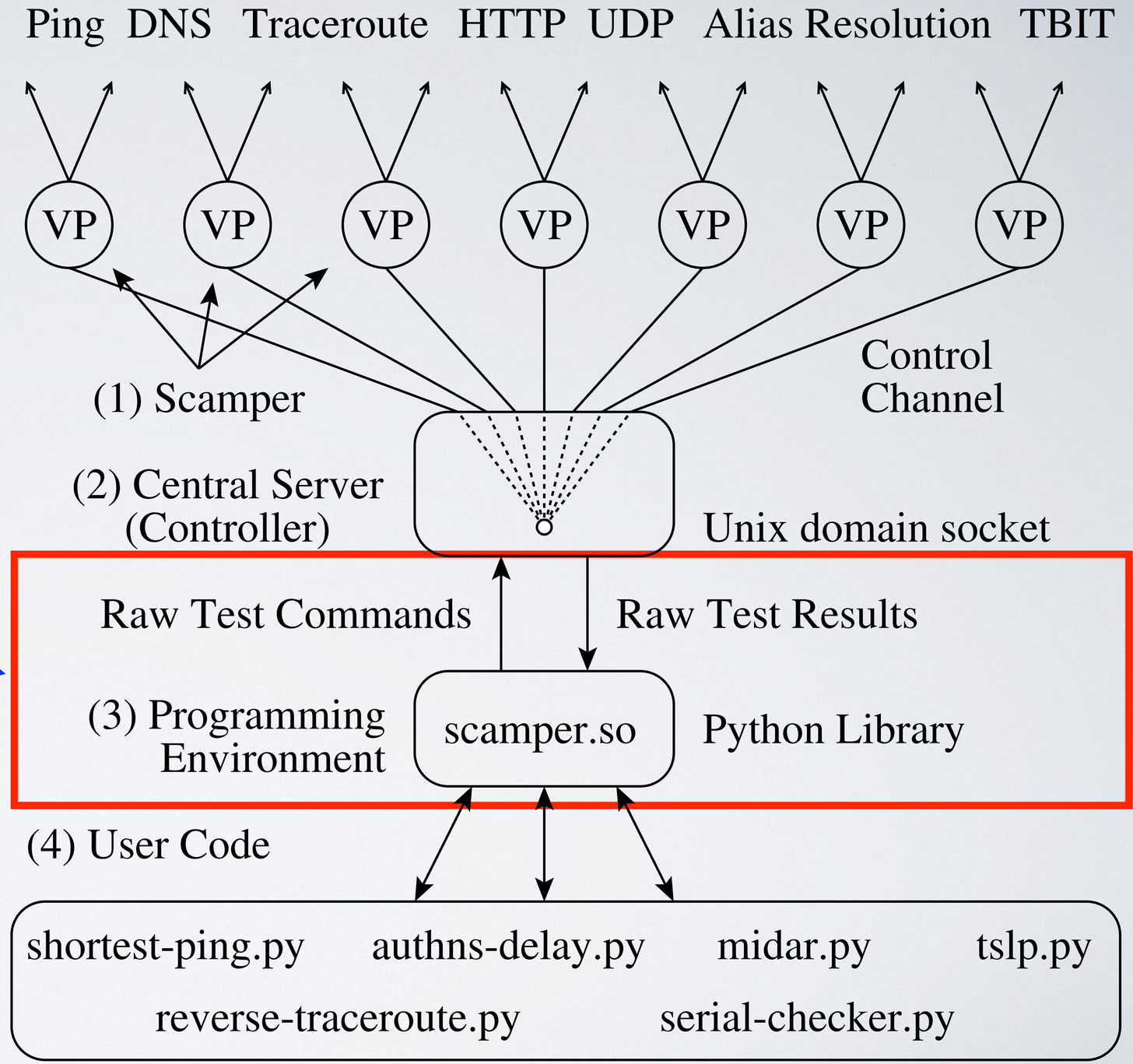
We used scamper's remote controller to manage VPs, which provides a multiplexed (mux) interface to all VPs with a single socket.



System Architecture

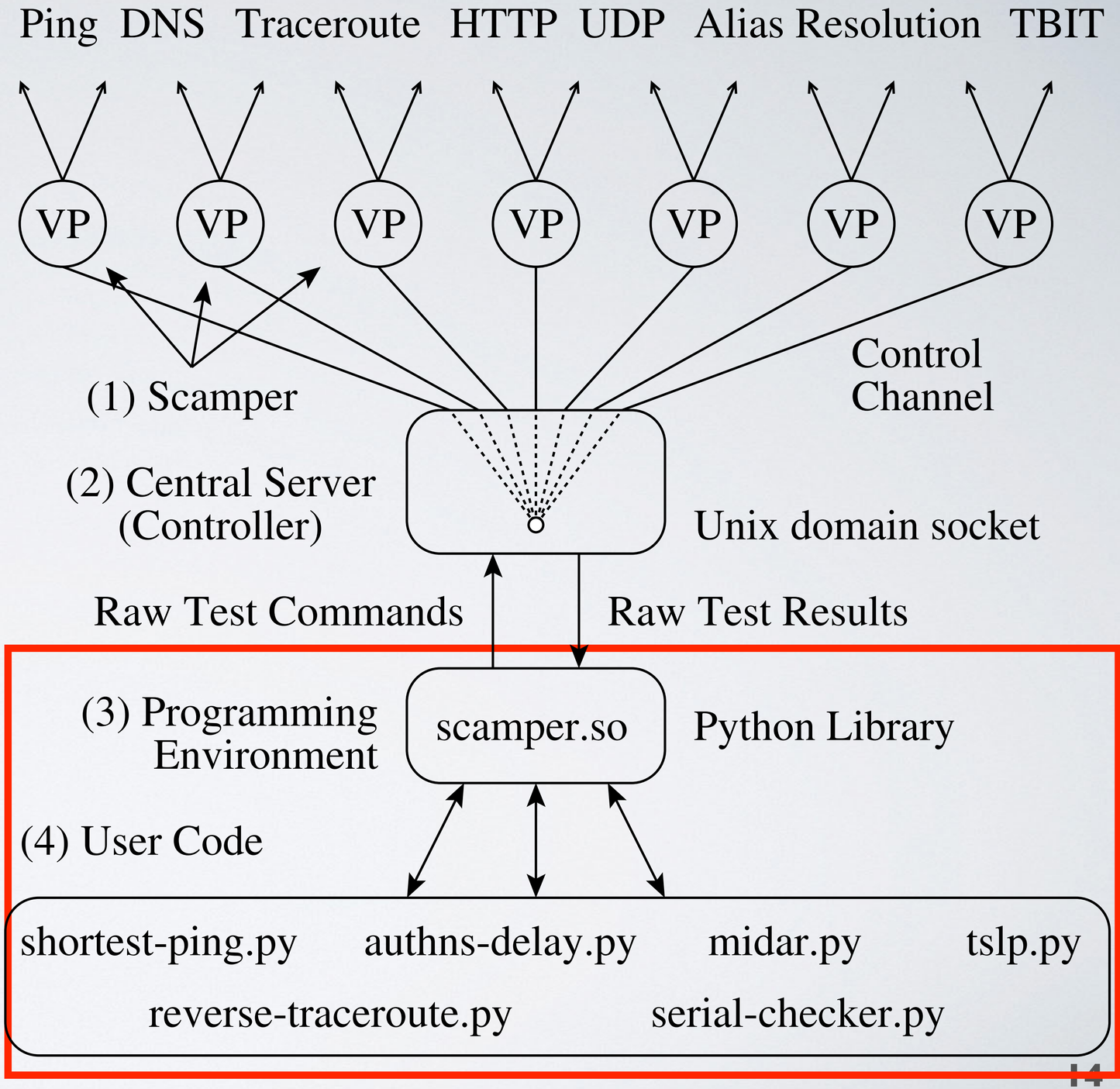
We built a Python module to interface with these VPs, providing methods to send measurement requests to VPs, and interpret the responses.

Module is written in ~12K lines of Cython



System Architecture

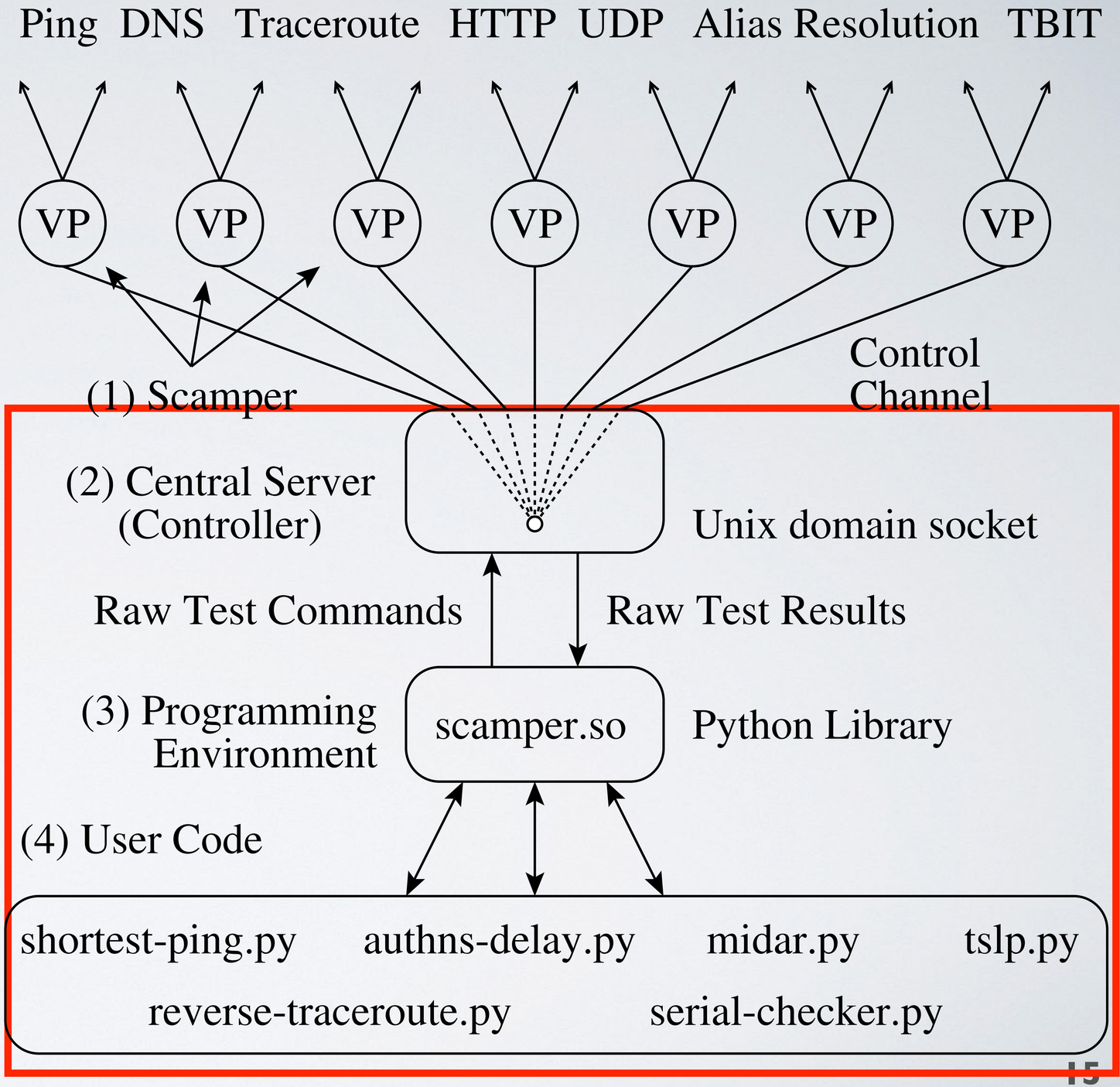
User code interfaces with the system's capabilities using the Python module



System Architecture

Currently, user code runs on the same system as the controller.

But it is possible to run user code in a VM or container separated from the controller.



First Example: List of VPs

```
$ python3 mux-vps.py /run/ark/mux
```

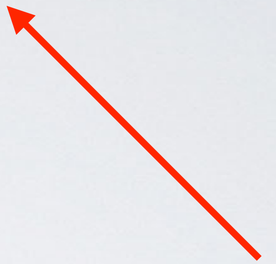
Location of mux socket

```
import sys
from scamper import ScamperCtrl

ctrl = ScamperCtrl(mux=sys.argv[1])
```

Many scripts will begin with code similar to this

First Example: List of VPs

```
$ python3 mux-vps.py /run/ark/mux
```

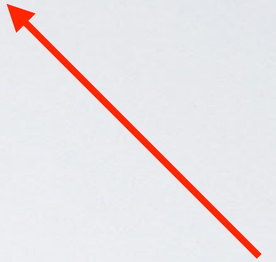


Location of mux socket

```
import sys
from scamper import ScamperCtrl

ctrl = ScamperCtrl(mux=sys.argv[1])

for vp in ctrl.vps():
    print(vp.name)
```

Second Example: List of VPs in a Country

```
$ python3 mux-vps-cc.py /run/ark/mux NZ
```



Location of mux socket

```
import sys
from scamper import ScamperCtrl

ctrl = ScamperCtrl(mux=sys.argv[1])

vps  = [vp for vp in ctrl.vps() if vp.cc == sys.argv[2]]
for vp in vps:
    print(vp.name)
```

Synchronous Blocking: single ping

```
import sys
from scamper import ScamperCtrl

if len(sys.argv) != 3:
    print("usage: single-ping.py $mux $dst")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
```

Many scripts will begin with code similar to this

Synchronous Blocking: single ping

```
import sys
from scamper import ScamperCtrl

if len(sys.argv) != 3:
    print("usage: single-ping.py $mux $dst")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])
```

Many scripts will begin with code similar to this

Connect to one scamper instance.

Synchronous Blocking: single ping

```
import sys
from scamper import ScamperCtrl

if len(sys.argv) != 3:
    print("usage: single-ping.py $mux $dst")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])
ping = ctrl.do_ping(sys.argv[2], sync=True)
```

Many scripts will begin with code similar to this

Connect to one scamper instance.

Issue ping synchronously

Synchronous Blocking: single ping

```
import sys
from scamper import ScamperCtrl

if len(sys.argv) != 3:
    print("usage: single-ping.py $mux $dst")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])
ping = ctrl.do_ping(sys.argv[2], sync=True)
```

Many scripts will begin with code similar to this

Connect to one scamper instance.

Issue ping synchronously

Synchronous: Measurement must complete before script continues

Use in small-scale, single-VP, simple measurements

E.g. Single pings, traceroutes, DNS lookups, HTTP queries.

Synchronous Blocking: single ping

```
import sys
from scamper import ScamperCtrl

if len(sys.argv) != 3:
    print("usage: single-ping.py $mux $dst")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])
ping = ctrl.do_ping(sys.argv[2], sync=True)

if ping.min_rtt is not None:
    print(f"{ping.inst.name} {(ping.min_rtt.total_seconds()*1000):.1f} ms")
else:
    print(f"no responses for {sys.argv[2]}")
```

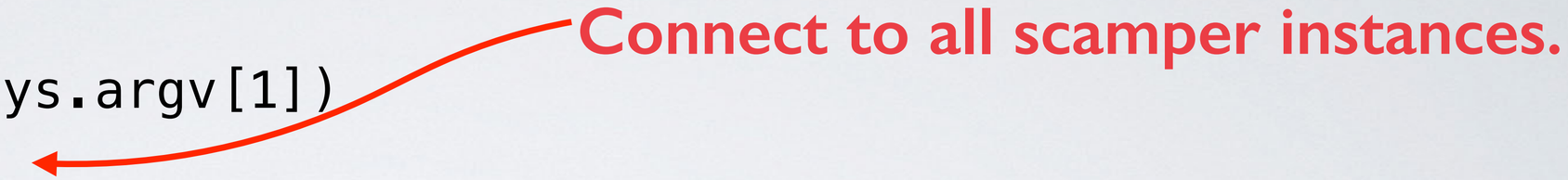
Many scripts will begin with code similar to this

Connect to one scamper instance.

Issue ping synchronously

Asynchronous Blocking: Shortest Ping

```
if len(sys.argv) != 3:
    print("usage: shortest-ping.py $mux $ip")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps())
```



Asynchronous Blocking: Shortest Ping

```
if len(sys.argv) != 3:
    print("usage: shortest-ping.py $mux $ip")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps())
ctrl.do_ping(sys.argv[2], inst=ctrl.instances())
```

Connect to all scamper instances.

Issue pings on all VPs, which run async

Asynchronous Blocking: Shortest Ping

```
if len(sys.argv) != 3:
    print("usage: shortest-ping.py $mux $ip")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps())
ctrl.do_ping(sys.argv[2], inst=ctrl.instances())
```

Connect to all scamper instances.

Issue pings on all VPs, which run async

Asynchronous: issue multiple measurements, stop to collect results.

Use in small/mid-scale, multi-VP reactive measurements.

E.g. Geolocating single IP address, measuring RTT of authoritative nameservers for a zone

Asynchronous Blocking: Shortest Ping

```
if len(sys.argv) != 3:
    print("usage: shortest-ping.py $mux $ip")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps())
ctrl.do_ping(sys.argv[2], inst=ctrl.instances())

min_rtt = None
min_vp = None
for o in ctrl.responses(timeout=timedelta(seconds=10)):
```

Connect to all scamper instances.

Issue pings on all VPs, which run async

Block for responses, up to 10s

Asynchronous Blocking: Shortest Ping

```
if len(sys.argv) != 3:
    print("usage: shortest-ping.py $mux $ip")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps())
ctrl.do_ping(sys.argv[2], inst=ctrl.instances())

min_rtt = None
min_vp = None
for o in ctrl.responses(timeout=timedelta(seconds=10)):
    if o.min_rtt is not None and (min_rtt is None or min_rtt > o.min_rtt):
        min_rtt = o.min_rtt
        min_vp = o.inst

if min_rtt is not None:
    print(f"{min_vp.name} {(min_rtt.total_seconds()*1000):.1f} ms")
else:
    print(f"no responses for {sys.argv[2]}")
```

Connect to all scamper instances.

Issue pings on all VPs, which run async

Block for responses, up to 10s

Determine VP with minimum RTT

Mixture of Blocking: Nameserver Delay

```
if len(sys.argv) != 3:
    print("usage: authns-delay.py $mux $zone")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])

# get the list of NS for the zone
o = ctrl.do_dns(sys.argv[2], qtype='NS', wait_timeout=1, sync=True)
```

Synchronous Blocking query to get NSes

Mixture of Blocking: Nameserver Delay

```
if len(sys.argv) != 3:
    print("usage: authns-delay.py $mux $zone")
    sys.exit(-1)

ctrl = ScamperCtrl(mux=sys.argv[1])
ctrl.add_vps(ctrl.vps()[0])

# get the list of NS for the zone
o = ctrl.do_dns(sys.argv[2], qtype='NS', wait_timeout=1, sync=True)

# issue queries for the IP addresses of the authoritative servers
for ns in set(o.ans_nses()):
    ctrl.do_dns(ns, qtype='A', wait_timeout=1)
    ctrl.do_dns(ns, qtype='AAAA', wait_timeout=1)
```

Synchronous Blocking query to get NSes

Asynchronous Queries for NS IPs

Mixture of Blocking: Nameserver Delay

```
# collect the unique addresses
addr = {}
for o in ctrl.responses(timeout=10):
    for a in o.ans_addrs():
        addr[a] = o.qname
```

Build dict mapping IP to NS name

Mixture of Blocking: Nameserver Delay

```
# collect the unique addresses
addr = {}
for o in ctrl.responses(timeout=10):
    for a in o.ans_addrs():
        addr[a] = o.qname

# collect RTTs for the unique IP addresses
for a in addr:
    ctrl.do_ping(a)
rtts = {}
for o in ctrl.responses(timeout=10):
    rtts[o.dst] = o.min_rtt
```

Build dict mapping IP to NS name

Conduct async pings to all IPs, collect results

Mixture of Blocking: Nameserver Delay

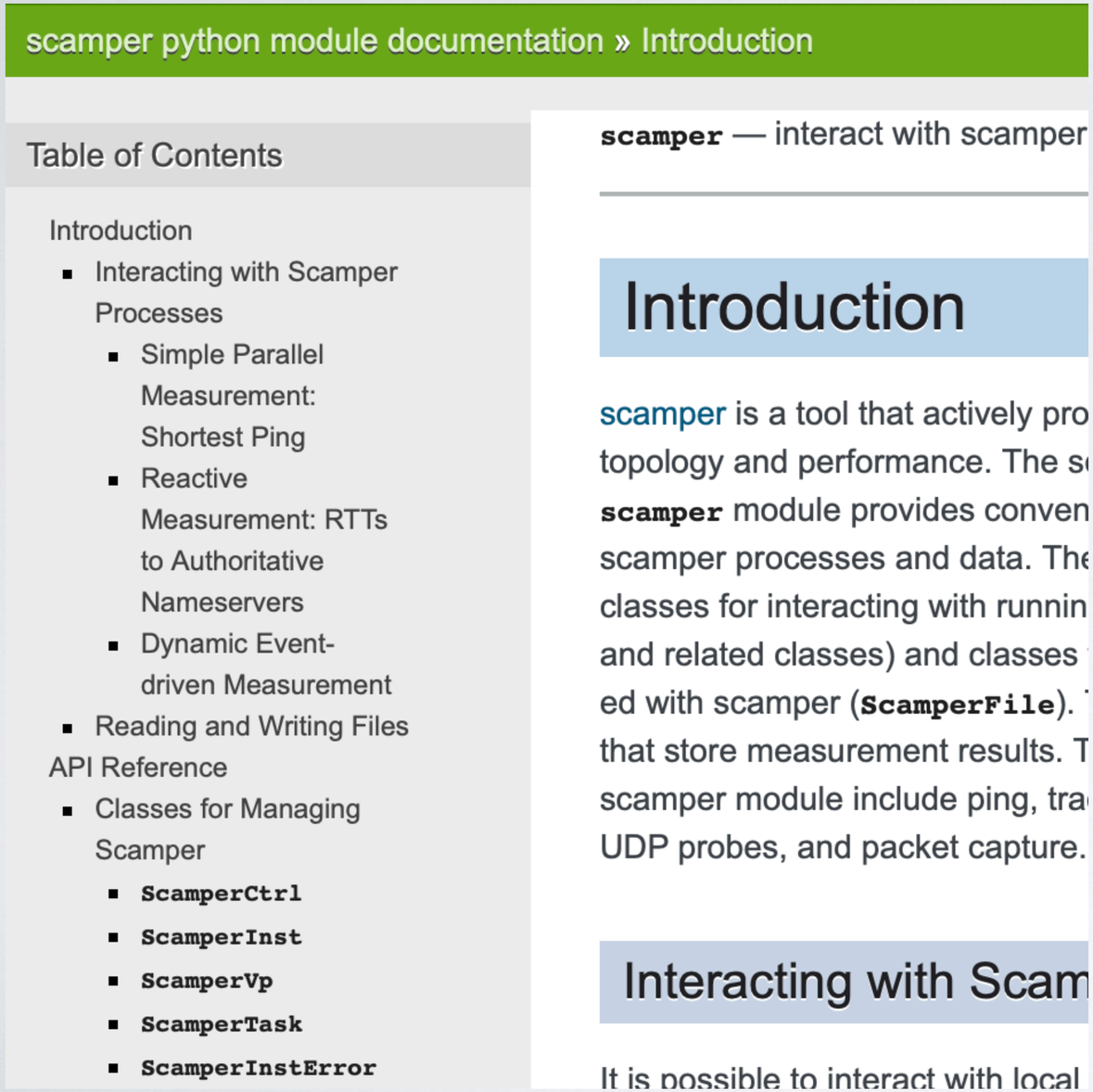
```
# collect the unique addresses
addr = {}
for o in ctrl.responses(timeout=10):
    for a in o.ans_addrs():
        addr[a] = o.qname

# collect RTTs for the unique IP addresses
for a in addr:
    ctrl.do_ping(a)
rtts = {}
for o in ctrl.responses(timeout=10):
    rtts[o.dst] = o.min_rtt

# print authns by delay
for a in dict(sorted(rtts.items(), key=cmp_to_key(rttcmp))):
    print(f"{addr[a]} {a} " +
          (f"{(rtts[a].total_seconds() * 1000):.1f}"
           if rtts[a] is not None else "???"))
```

Build dict mapping IP to NS name

Conduct async pings to all IPs, collect results

Print NS sorted by RTT

Documentation



- Environment is thoroughly documented at https://www.caida.org/catalog/software/scamper/python/
- We provided the documentation to LLMs
  - asked the LLM to write measurement scripts.
  - they do a reasonable job, but scripts often need tweaking.

Summary

- We built an integrated active measurement programming environment, where we provide a programming environment that exposes measurement capabilities on distributed vantage points
- We recently held a successful hackathon the weekend before AIMS 2025 where participants built scripts to collect and analyze data across different topics (topology, DNS, web)
- We invite researchers to use our platform. ark-info@caida.org

https://www.caida.org/catalog/software/scamper/python/

Acknowledgments

- This work was supported by NSF grants OAC-2131987, CNS-2120399, CNS-2323219, and CNS-2212241.

- This paper represents only the position of the authors

An Integrated Active Measurement Programming Environment

A design for a next-generation Internet Measurement Infrastructure

Matthew Luckie

Brendon Jones

Shivani Hariprasad

Ken Keys

k claffy

Raffaele Sommese

Ricky Mok

PAM 2025

caida

Domain Specific Language: Pros + Cons

- ☑ High-level abstractions for common measurements that have multiple complicated steps
- ☑ Easier for non-programmers
- ☑ Implementation of primitive is probably better than a roll-your-own
- ☑ Harder to do bad things; send bad packets, too many packets, write to local disk, consume CPU
- ☑ Can describe capabilities to VP hosts

Domain Specific Language: Pros + Cons

- ☑ High-level abstractions for common measurements that have multiple complicated steps
- ☑ Easier for non-programmers
- ☑ Implementation of primitive is probably better than a roll-your-own
- ☑ Harder to do bad things; send bad packets, too many packets, write to local disk, consume CPU
- ☑ Can describe capabilities to VP hosts

- ☐ DSL still needs to be learned
- ☐ Base language (Python, Lua, Perl, Ruby) might not be well-known
- ☐ Limited scope / cut down; might make it difficult to do some types of tasks
- ☐ Researchers reliant on DSL maintainers exposing useful primitives and logic
- ☐ Researchers reliant on DSL maintainers keeping it up to date to work on modern systems, modern protocols

Approaches to managing measurements

Approach	Summary	When to use	Examples
Synchronous Blocking	measurement must complete before script continues	Small-scale, single-VP, simple measurements	Single pings, traceroutes, DNS lookups, HTTP queries, etc.

Approaches to managing measurements

Approach	Summary	When to use	Examples
Synchronous Blocking	measurement must complete before script continues	Small-scale, single-VP, simple measurements	Single pings, traceroutes, DNS lookups, HTTP queries, etc.
Asynchronous Blocking (default)	issue multiple measurements, stop to collect results	Small/Mid-scale, multi-VP reactive measurements	Geolocating single IP address, measuring RTT of authoritative nameservers for a zone

Approaches to managing measurements

Approach	Summary	When to use	Examples
Synchronous Blocking	measurement must complete before script continues	Small-scale, single-VP, simple measurements	Single pings, traceroutes, DNS lookups, HTTP queries, etc.
Asynchronous Blocking (default)	issue multiple measurements, stop to collect results	Small/Mid-scale, multi-VP reactive measurements	Geolocating single IP address, measuring RTT of authoritative nameservers for a zone
Asynchronous Non-Blocking	event-driven streaming of measurements, reacting to results	Internet-scale measurement	Geolocating millions of IP addresses, Internet-scale Alias resolution, Trufflehunter