Path Transparency What is it, and why should we care?

Mirja Kühlewind <mirja.kuehlewind@ericsson.com>
TMA PhD School, June 18, 2019, Paris

This work was performed within the H2020 mami project. See https://mami-project.eu

Preparation for Part II

Install **git**, **vagrant**, and **virtualbox** and:

```
git clone https://github.com/mami-project/pathspider-MNM19.git
```

Note: This is a private repo with specific configurations for this tutorial. The VM image in this repo is preconfigured to use a university VPN server in order to avoid problems with the local network. This VPN is only available during the tutorial. Please do not share this repo further, but the use the official PATHspider in future instead:

https://github.com/mami-project/pathspider

Overview Part I

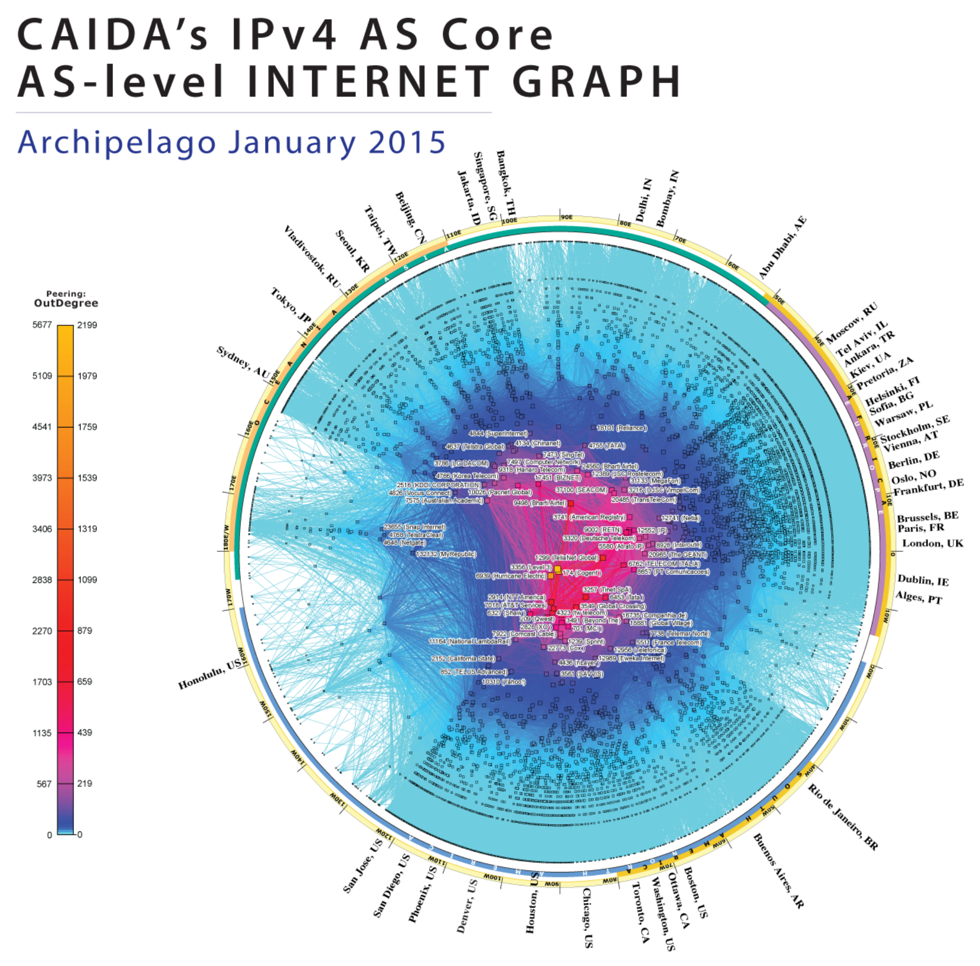
- **Why measure the Internet and how?**
  - Internet Measurement Architecture
  - Tools for Active Measurement
- **What is Path Transparency?**
  - PATHspider as an active measurement tool
  - Some Examples and Results
- **What to do with all the Measurement Data?**
  - Path Transparency Observatory

Overview Part II

- Hands-on work with PATHspider
  - Let’s start some measurements!
  - Target Lists using Hellfire
  - PATHspider Plugins
  - Use of scapy for packet forging
  - Observations and Analysis

Why measure the Internet?

Leonardo Rizzi CC-BY-SA



- **Operations**: keep the Internet working
  - "What's broken?"
  - "Who's attacking me?"
  - "Are things working as expected; if not, why not?"
  - "How should we plan future expansion of our network?"
- **Research:** understand the Internet as a phenomenon in itself
  - "What does the network look like?"
  - "How will the network look tomorrow?"
  - "Hm, that's interesting, what's that?"
- **Engineering:** support protocol design decisions with data

(Path transparency is primarily motived by the third, but the techniques we'll explore today are applicable to all three activities)

Privacy and Good Practise

- "Strategies for sound internet measurement", Vern Paxson, 2004. In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement (IMC '04).
  - http://dx.doi.org/10.1145/1028788.1028824
  - Calibration and meta data
  - Reproducibility, handling of large data set, and making data publicly available
- Aggreation, Minimization, and IP Address Anonymization to address Privacy Risks
  - https://datatracker.ietf.org/doc/draft-learmonth-pearg-safe-internet-measurement/
  - Tor Metrics (https://metrics.torproject.org): *"Analyzing a live anonymity system must be performed with great care so that the users' privacy is not put at risk. Any metrics collected must not undermine the anonymity or security properties of the Tor network."*

Active versus Passive Measurement

- **Active measurement** uses *dedicated measurement traffic* to induce a measurable reaction from the network and/or far endpoint.
    - Examples: ping, traceroute, everything RIPE Atlas does
    - Tradeoffs: you can measure what you're aiming at, but (1) you might not be measuring exactly what productive traffic sees and (2) you have to pay the overhead of unproductive measurement traffic.
- **Passive measurement** observes *productive* traffic and draws inferences about the state of the network and endpoints from what it can see.
    - Example: IPFIX, wireshark
    - Tradeoffs: what you observe is what you get, but you can only choose paths opportunistically, and you have to be *very* careful not to over-observe (end-user privacy).

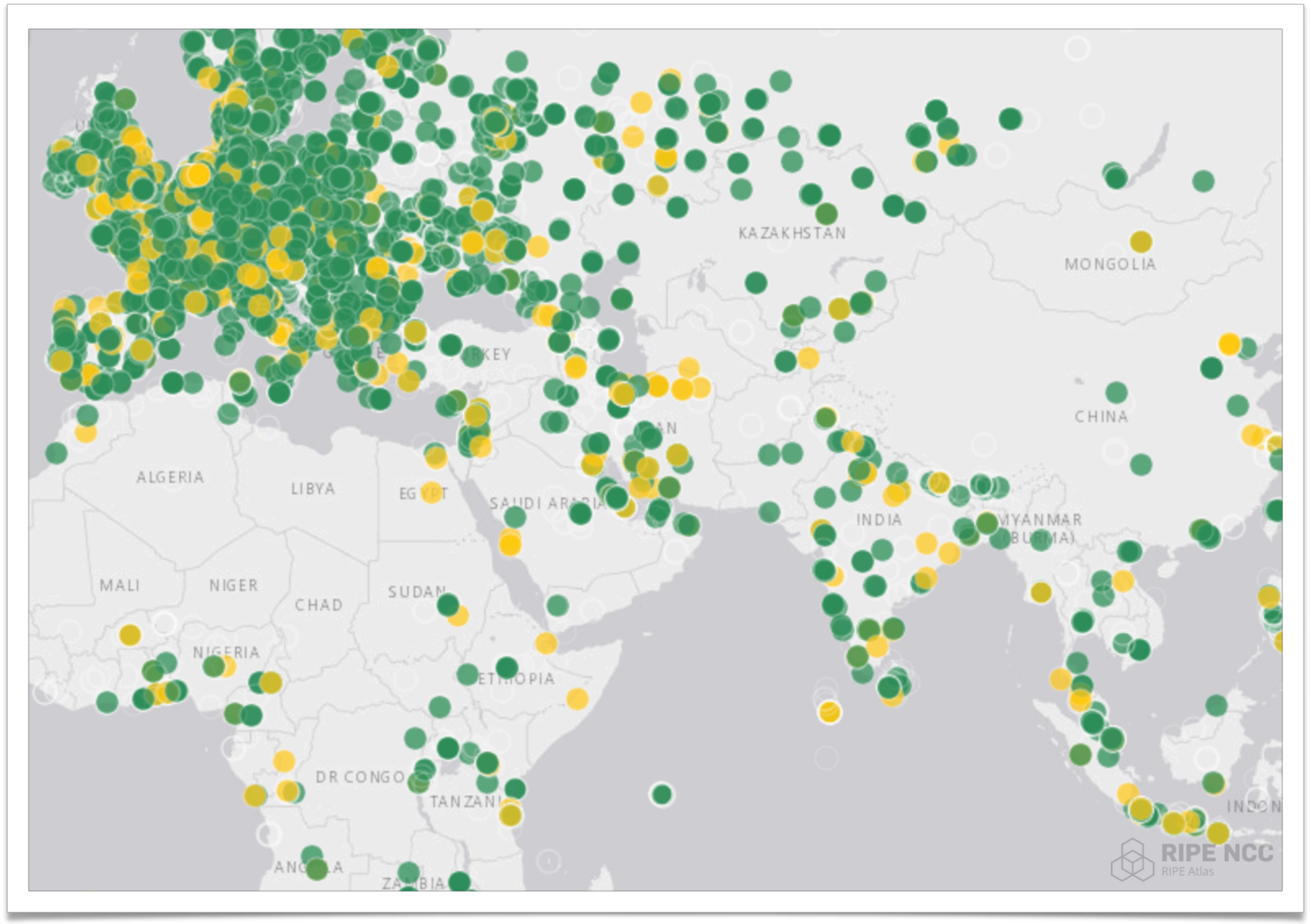
(We focus today on active measurement.)

Context in Active Measurements: tradeoffs in selecting vantage points

- Increasing choice in vantage points:
  - Active measurement networks (e.g. RIPE Atlas); including residential measurements (e.g. mLab) and mobile testbeds (e.g. MONROE)
  - Mesh of testbed nodes (e.g. Planetlab, Ark)
  - Core endpoints (e.g. DigitalOcean, AWS) toward public/uncontrolled targets (e.g. Alexa TopN)
  - Crowdsourcing can provide targeted endpoint with specific characteristics

Measurement Networks (e.g. RIPE Atlas)

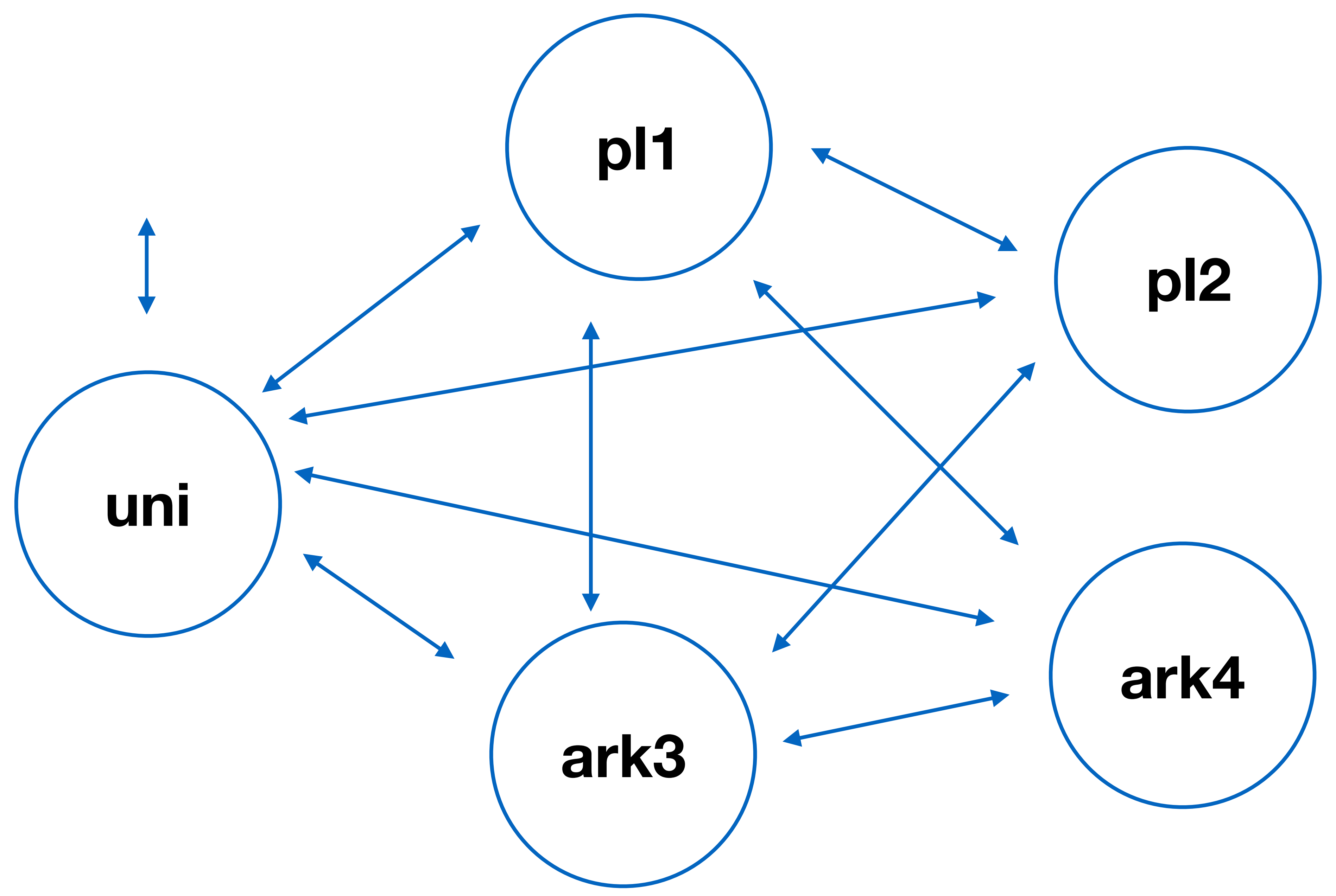
- Many (thousands) of small probes hosted on (often volunteer) networks allow shared access to simple active measurement infrastructure
- Advantages: scale, ease of use, diversity in network access
- Disadvantage: limits on the kinds and scale of measurements that can be done.



Testbed Mesh Measurements (e.g. PlanetLab)

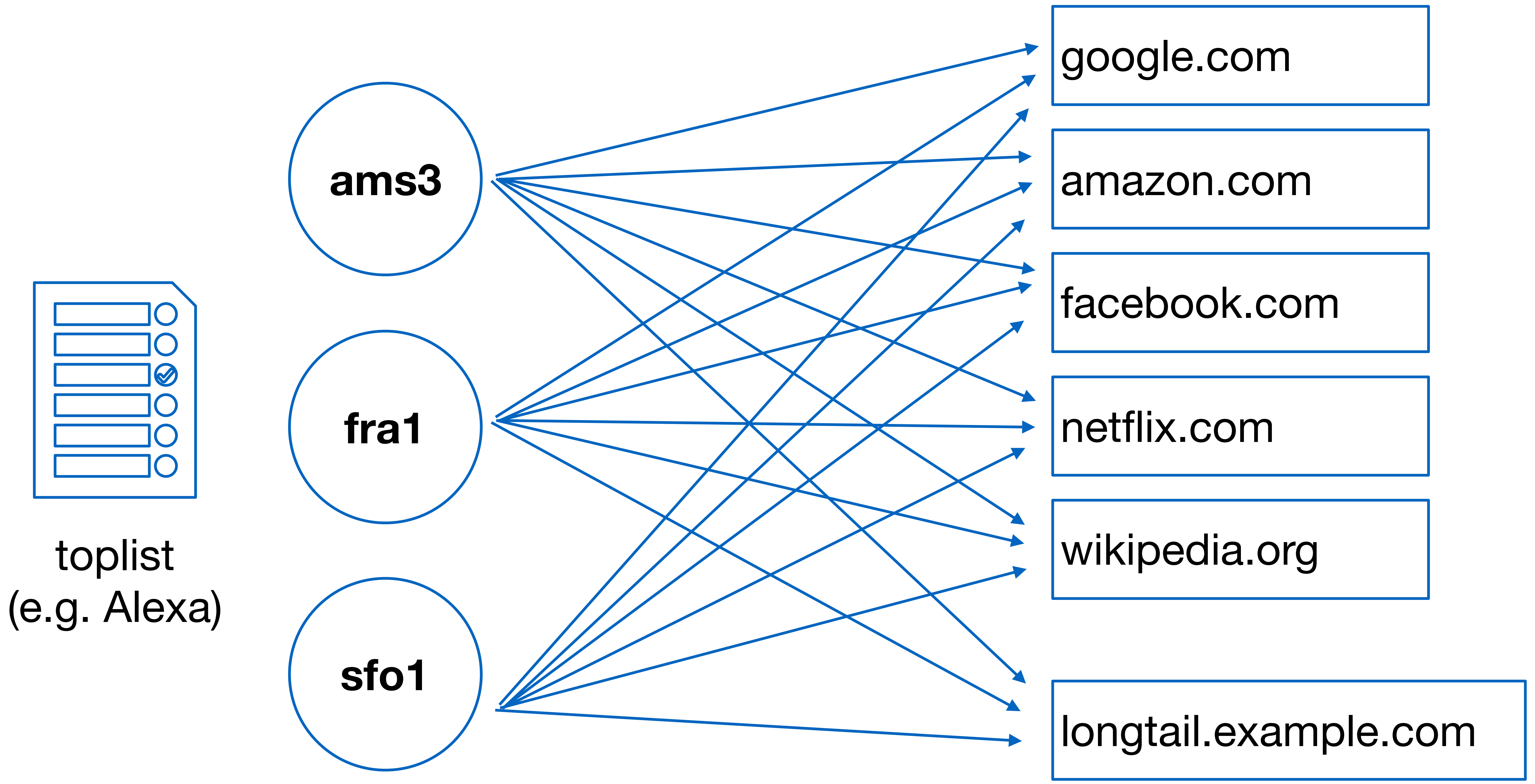
- Advantage: full control of both endpoints (path isolation and two-way tracing are possible)

- Disadvantage: hard to set up, testbed networks tend to be non-representative.



Core to Public Measurements

- Generate a list of public targets and measure it from multiple vantage points
- Advantage: the cloud scales nicely
- Disadvantages: one-sided measurement, inference required based on protocol behavior, toplist bias, not useful for access network behavior.



Crowdsourcing

- Pay people to run small measurement tools from their own Internet connections.
- Advantages: large diversity of access network connectivity.
- Disadvantages: usually non-expert workers, so tooling needs to be packaged to be easy to use in very diverse endpoint environments, bias difficult to quantify.

Tools for active measurements

- Community uses a limited set of powerful tools

  or

- "let me hack up this quick little script"?

- Using a common set of tools makes measurements more comparable and reproducible
  - I don't need to get your toolchain up and running to run your measurement on my network
  - When we all use the same tools, we all understand the limitations

In the beginning, there was ping, and it was good.

- Send an ICMP Echo Request, expect a corresponding ICMP Echo Reply.
  - To measure* two-way latency: $RTT_{ping} = t_{recv} - t_{send}$
  - To measure loss** $RTT_{ping} \cong \infty \rightarrow$ lost

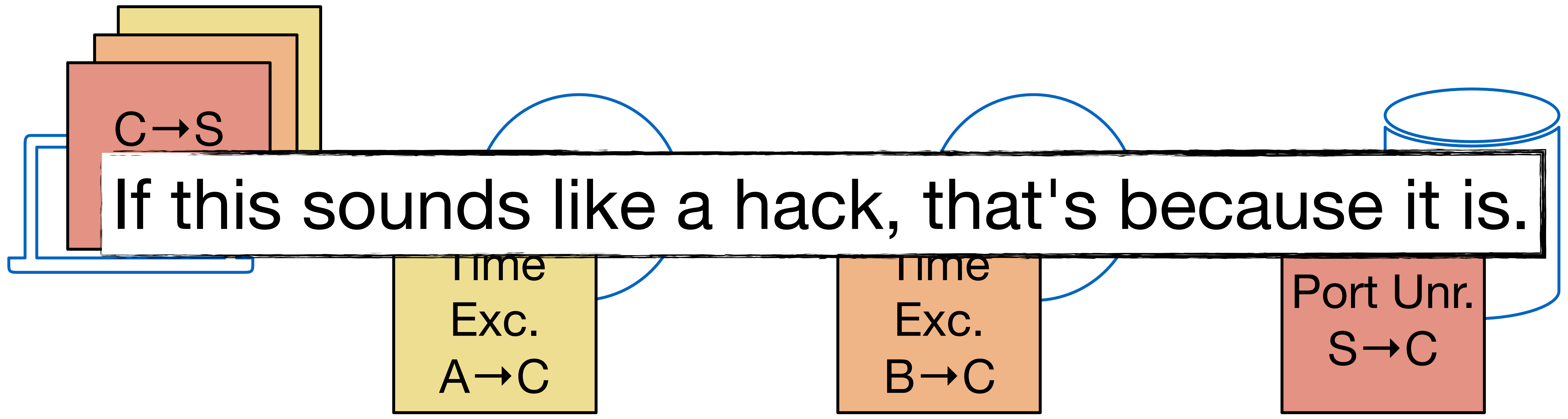
echo request
server
echo reply

*Might not measure what you want: ICMP traffic may be handled differently from productive traffic; ICMP terminated by the kernel, so doesn't correspond to application status.

**No way to split forward from reverse path: bad for localizing faults.

traceroute: making educated guesses about where the packets are going

- Send (ICMP, UDP, TCP) packets with a TTL lower than the number of hops between the source and the destination.
  - For TTL *n*, the *n*th hop* away from the source should** send back an ICMP Time Exceeded message.
  - Do this iteratively to make a list*** of hops on the forward path**** with RTTs.



If this sounds like a hack, that's because it is.

* where "hop" means "a router that speaks IP and decrements TTL"; tunnels are invisible

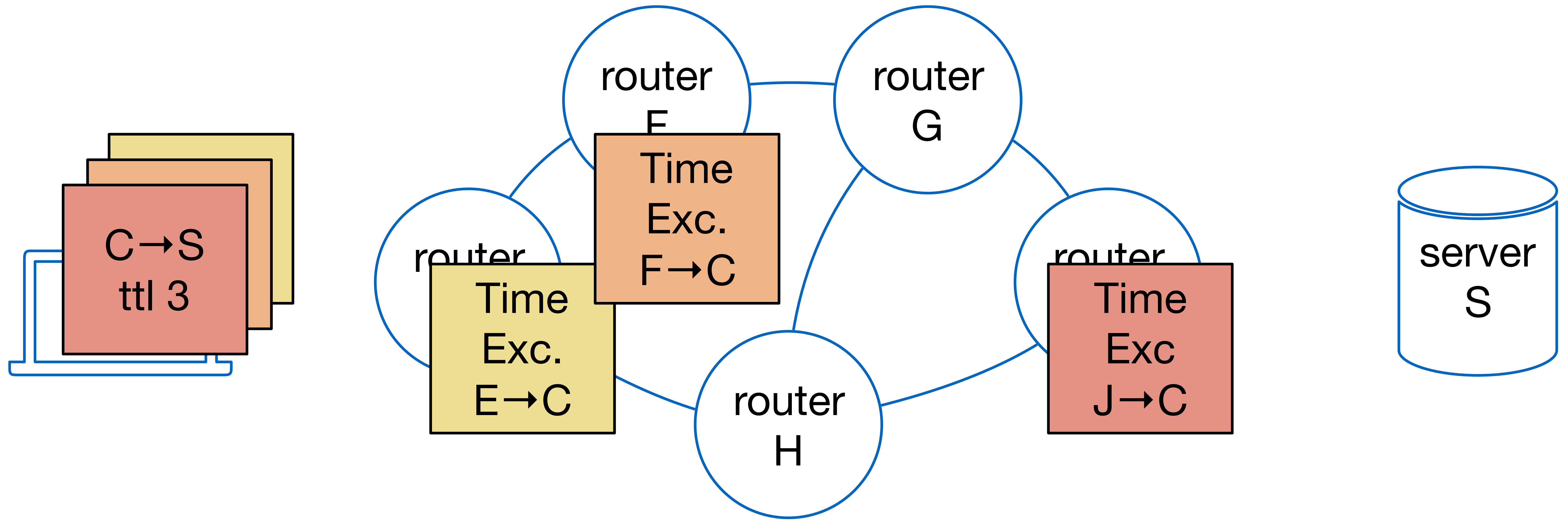
** ICMP is considered by some to be "reconnaissance activity", and therefore blocked at network borders

*** this list may or may not have any relationship to actual connectivity between routers, as a router may use any of its addresses to reply from, and using parts of the flow key to identify different hops will cause traceroute packets to take (wildly) different paths in the presence of ECMP (see Paris Traceroute).

**** no way to split paths: only the forward path is visible (but see various Reverse Traceroute schemes)

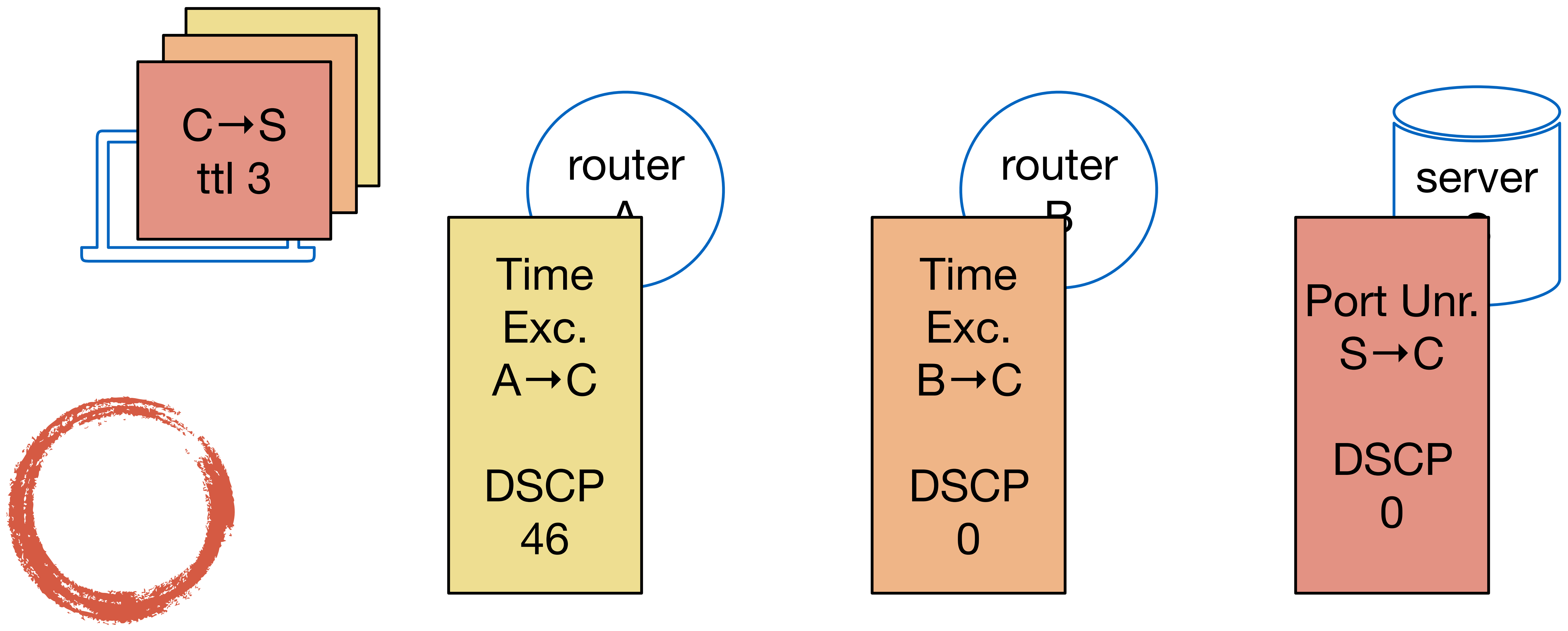
Paris traceroute: making even better guesses about where the packets are going

- Equal-cost multipath routing (ECMP) leads to multiple paths for the same source-destination pair...
  - ...hashing on fields that traditional Traceroute uses to identify hops



- Paris traceroute addresses this by holding flow identifiers constant.

Tracebox: getting the path to tell you how it's messing with your packets



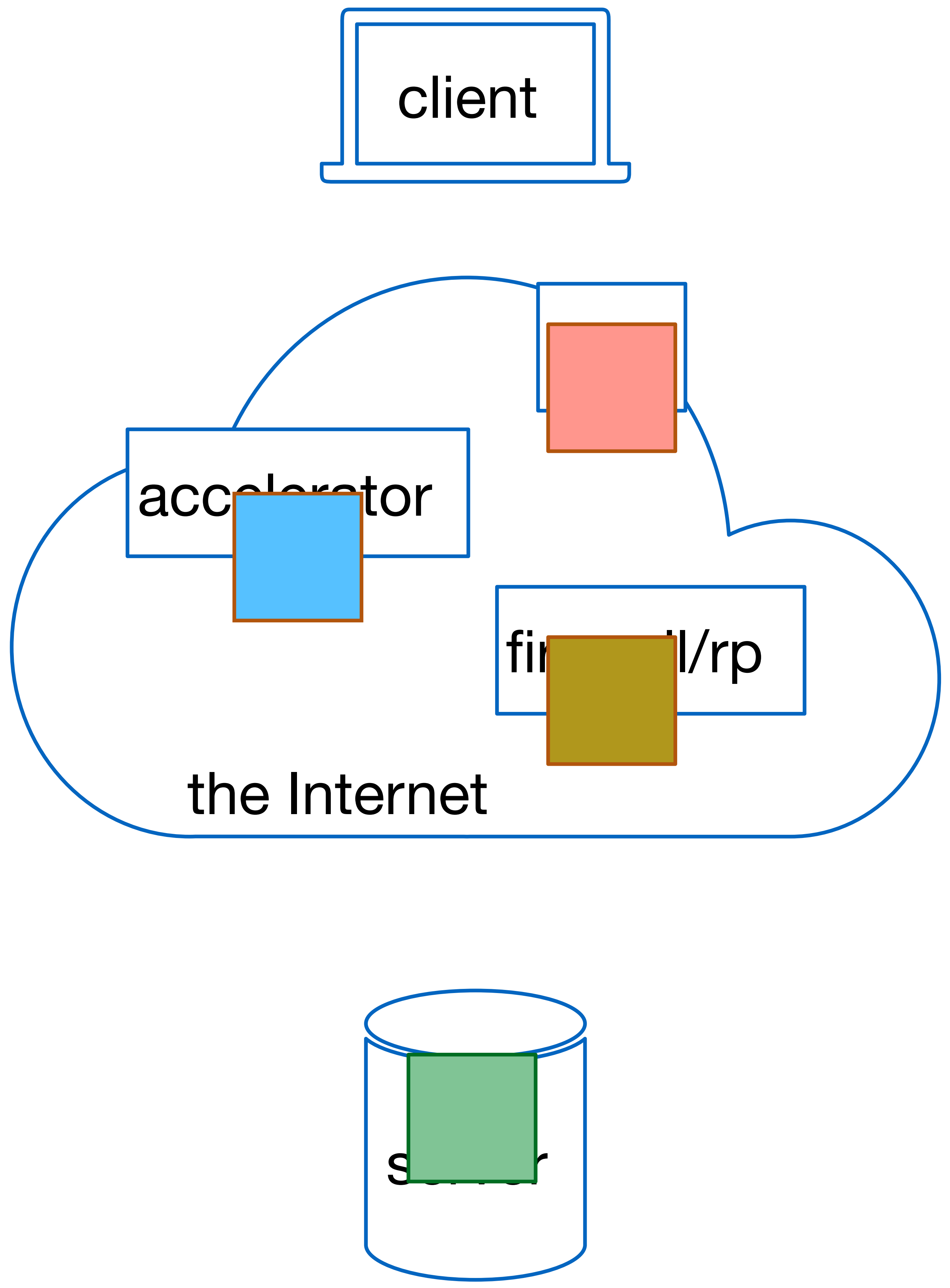
Path transparency measurement (a specific active measurement task)

- *End-to-end principle*: a maximally capable network made of smart endpoints connected by dumb pipes

➡ A path is ***transparent*** if packets come out the other end of the pipe unchanged

- This is not how things actually are, especially at layer 4:
    - Network Address Translation (NAT)
    - Extension and TCP option blocking and stripping
    - TCP ACK/SEQ rewriting
    - etc, etc, etc, etc…

➡ Middlebox functions can impair the connectivity and treatment of end-to-end traffic



Why measuring Path Transparency?

- More data about the ***nature*** and ***quantitive distribution*** of path impairments need as input for the design of protocols and protocol extensions that can deal with interference

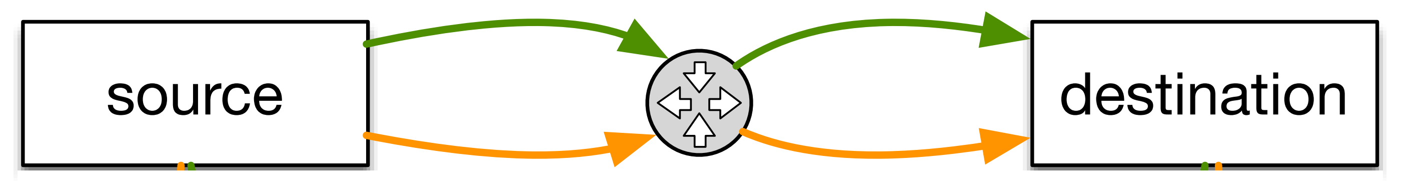
  ➡***Framework*** for measuring various different path impairments

- Combining disparate measurements leads to better insight
  - e.g. own measurement data, traceroutes, BGP, traces

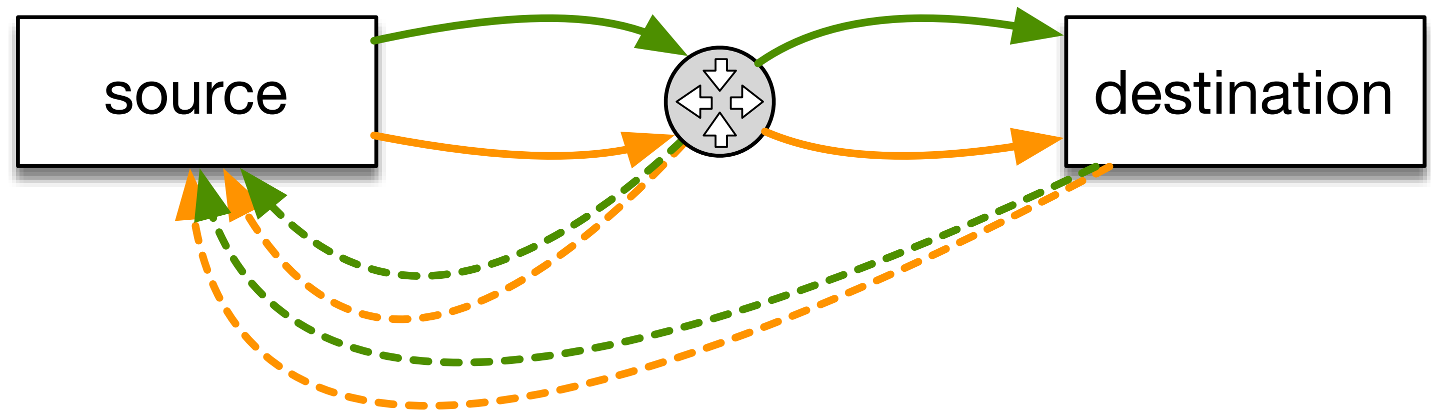
  ➡***Common data model*** for storage and analysis of path impairments

How to measure Path Transparency?

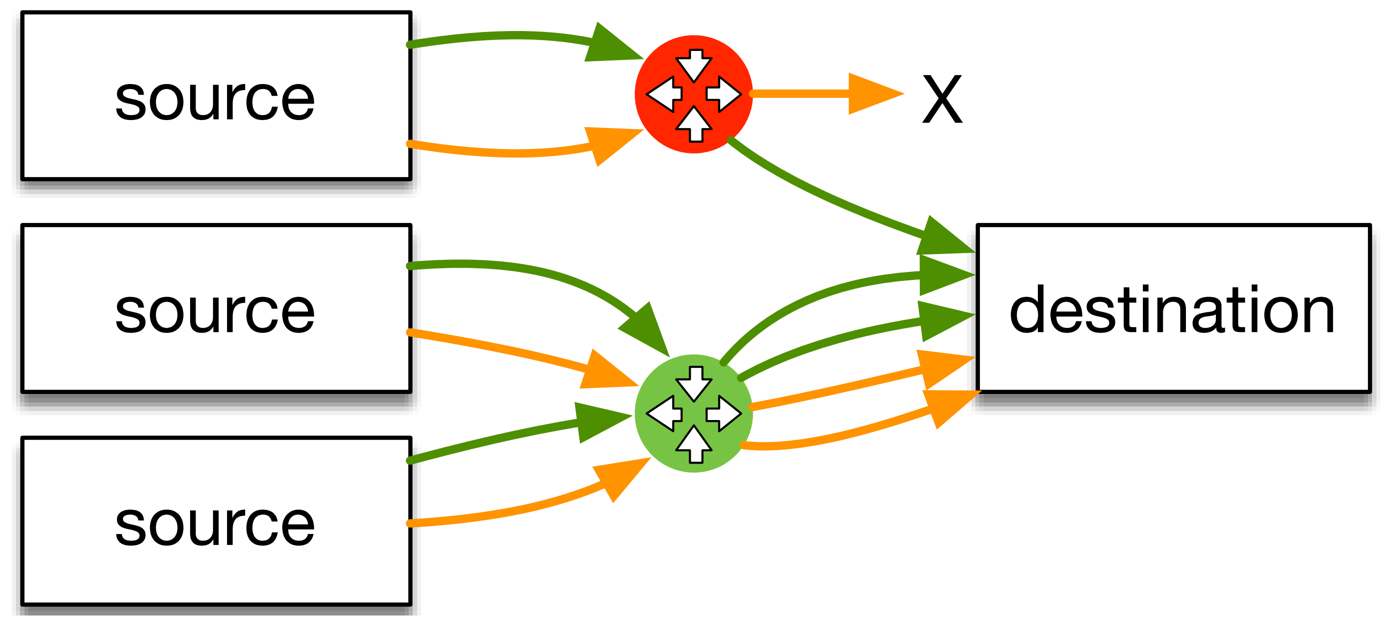
- Controlled experimentation (A/B testing)
  - Compare "vanilla" traffic to some feature under test



- Ideally, control both endpoints
  - Compare packets send to packets received



- To scale, infer path behavior from the destination's response
  - or induce routers to send us a response (traceroute)

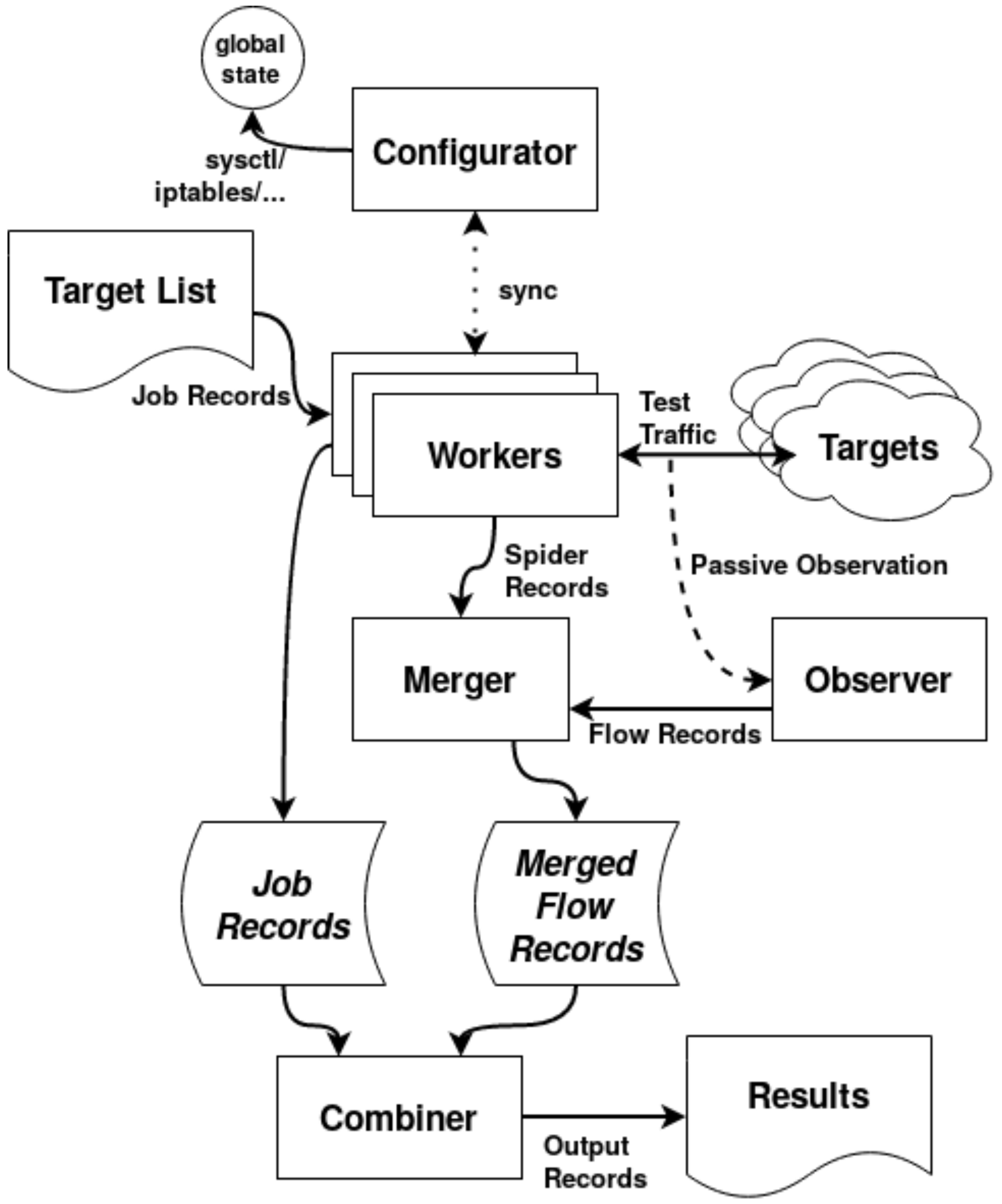


- Compare results from multiple vantage points
  - infer on-path versus on-endpoint or near-endpoint interference

PATHspider 2.0

https://pathspider.net/

- Generalized framework for A/B testing
- Plugin-based architecture with plugins available for
  - ECN connectivity and negotiation
  - DiffServ Codepoints
  - TCP Fast Open
  - …



PATHspider 2.0 new Features:

https://pathspider.net/

- Generalized to support more than just A/B testing
    - Any permutation of any number of tests now possible!
- PATHspider's is now using cURL for HTTP requests -> faster
- Framework for packet forging based plugins using Scapy
- Completely rewritten (in Go) target list resolver
    - Faster target list IP address resolution!
- Observer modules usable for standalone passive observation or analysis!

- See https://github.com/mami-project/pathspider/tree/2.0.0/
- Or https://pathspider.net

Path Transparency Observation

- An observation is an assertion that at a given **time,** a given **condition,** held on a given **path** (optionally associated with a value):

  ["0","2014-08-28T22:41:02Z","2014-08-28T22:46:25Z",
  "* 82.192.86.197","ecn.multipoint.connectivity.works","3"]

- A condition is some **state** of an **aspect** of a **feature** observed on a network e.g. ”**ecn.connectivity.works**” or ”**ecn.negotiation.reflected**”

  - **Feature:** what protocol or feature are we trying to use?
  - **Aspect:** what question are we asking about the feature?
  - **State:** what happens when we try to use it?
    - States are mutually exclusive for a given aspect on a path at a given time.

Paths

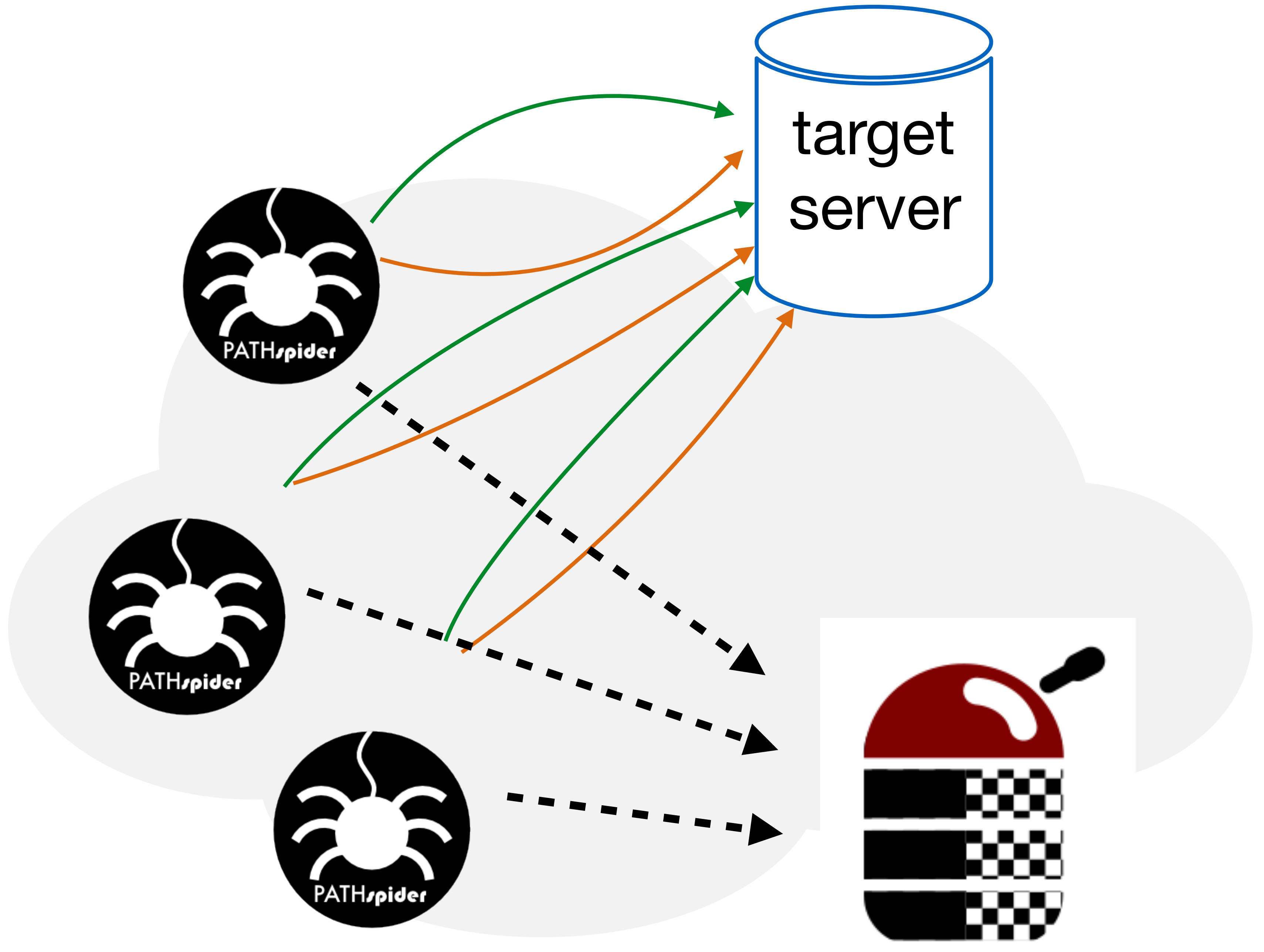
- A path is a **sequence of path elements** (IP addresses, prefixes, AS numbers, pseudonyms, or wildcard) on which an observation was taken
  - For active measurement: first path element is the **source** or device sending traffic, last path element is the **target** or device under test.

e.g. [digitalocean-ams3 * 104.24.127.228]

or [128.10.18.52 * 209.200.170.230 204.106.55.245 * 159.45.6.20]

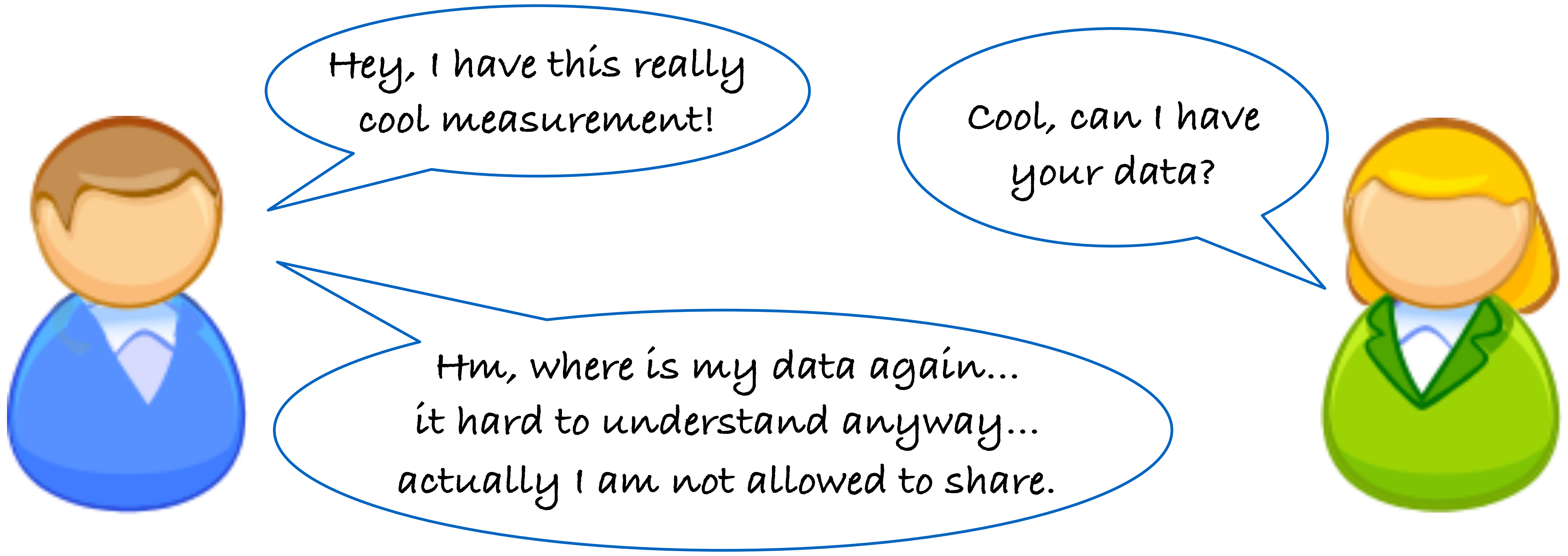
Scaling Path Transparency Measurement

- With PATHspider, we can do…
  - measurements from a single machine
  - in a single run
- How to compare measurements...
  - from multiple vantage points
  - across longer time scales?
- Centralize analysis in a ***Path Transparency Observatory***



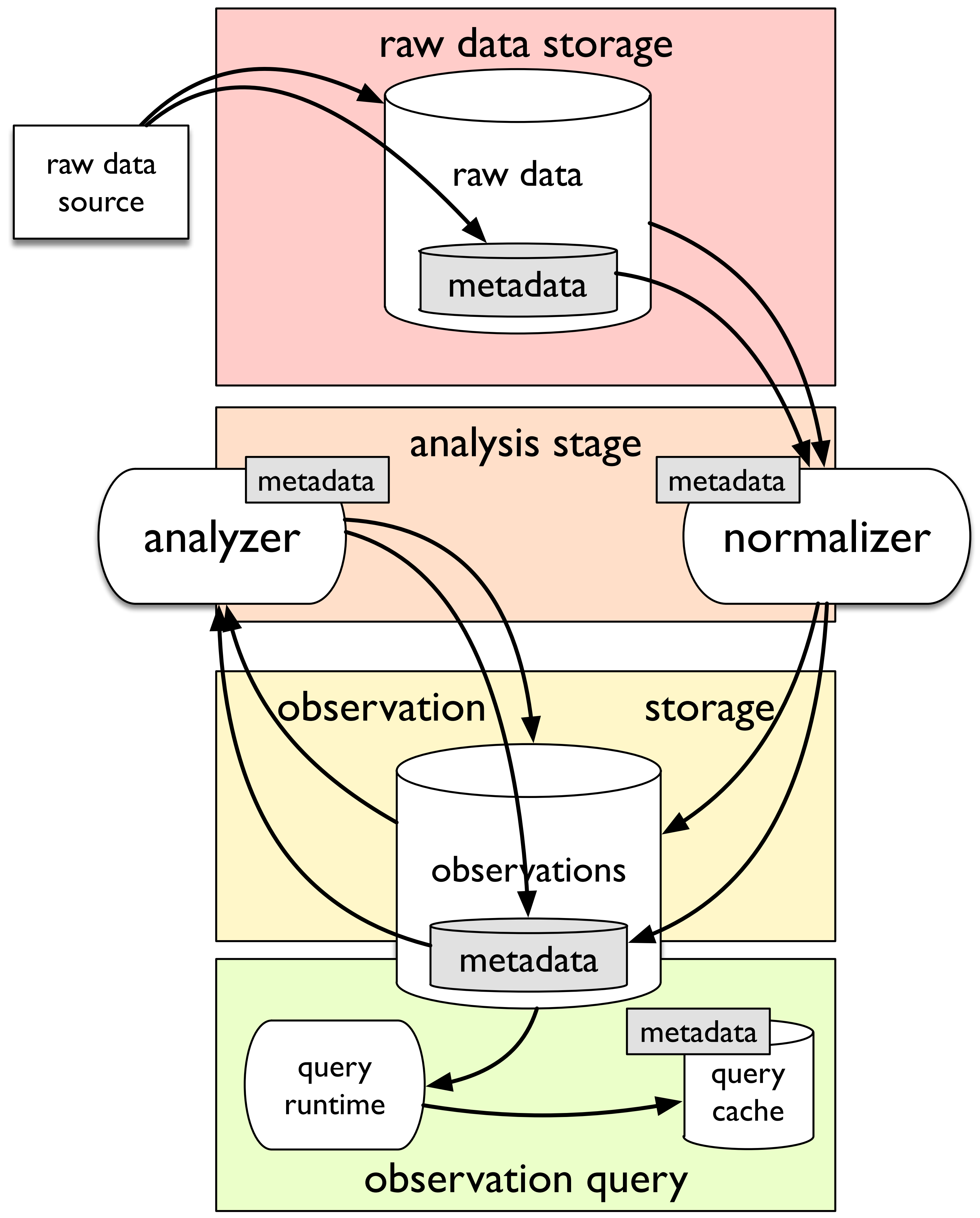
Role of an observatory

- Provides a central data repository to store data together with metadata, tracking provenance.
- Makes it easier to share intermediate and final results.


Hey, I have this really cool measurement!
Cool, can I have your data?
Hm, where is my data again... it hard to understand anyway... actually I am not allowed to share.

Design of the Path Transparency Observatory (PTO)

- ***Raw data*** submitted to the observatory
  …by value or reference
- **Normalizers** convert raw data to ***observations***, controlled by raw metadata
- **Analyzers** create derived/combined observations from (lower-level) observations
- ***Observations*** stored in a common schema, grouped into ***sets*** that share ***metadata***
- ***Queries*** over observations are always temporally scoped, and can be cached for permanent reference.

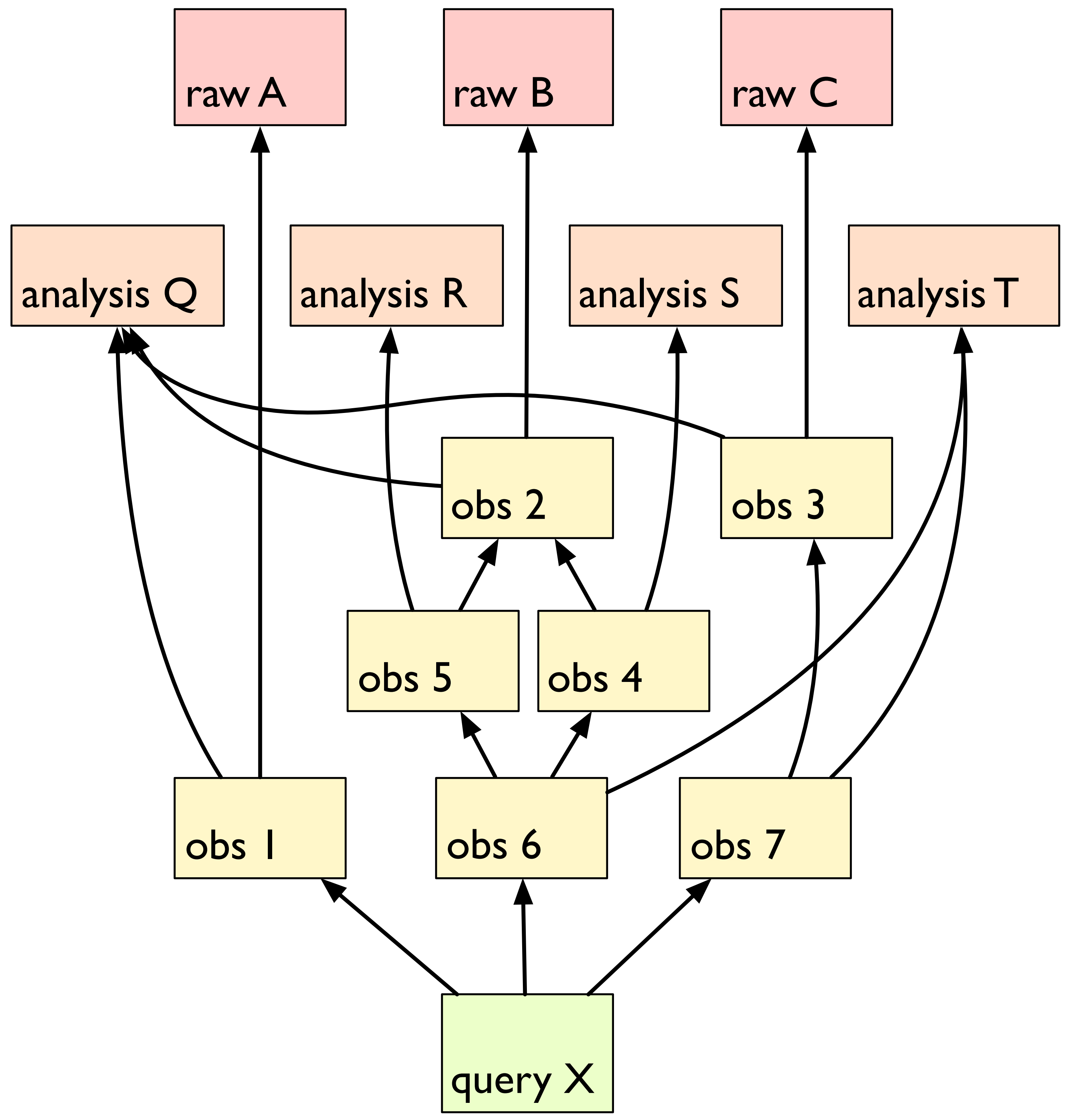


Design Goals of the PTO

Measurement data observatories can support better science through the following design goals:

- ***Comparability***: allow the results from diverse tools to be expressed in the same vocabulary so they can be compared.
- ***Repeatability***: keep enough metadata around so that future users of the data know how to repeat the experiment.
- ***Protection***: reduce information in raw data to only that needed for a particular analytical task.

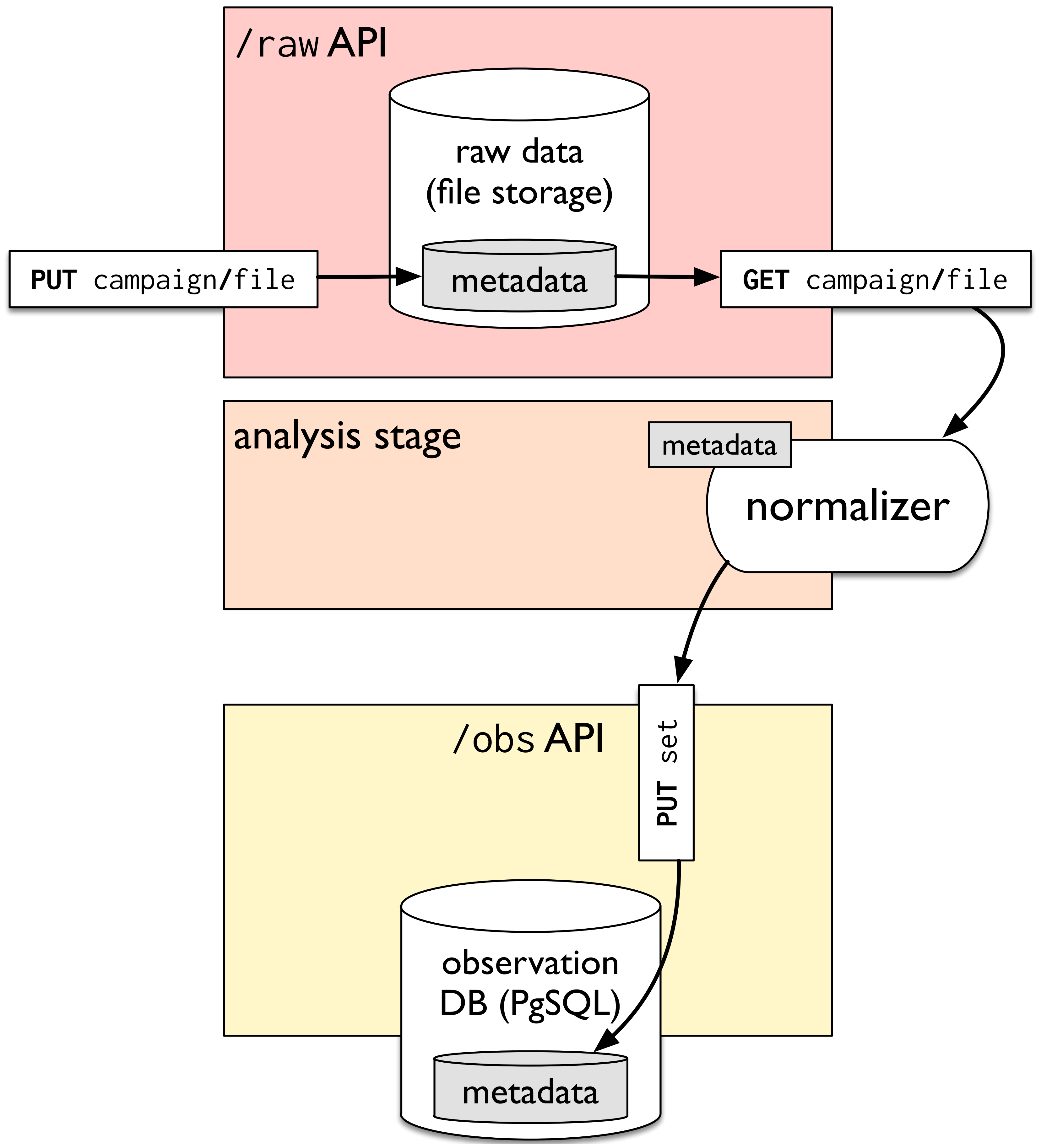
Supporting Repeatability: Provenance



- Every object refers to its antecedents
- Following these links back from a query or observation set results in a **provenance tree.**
- This provenance tree is, in effect, a set of instructions for recreating a given observation set.

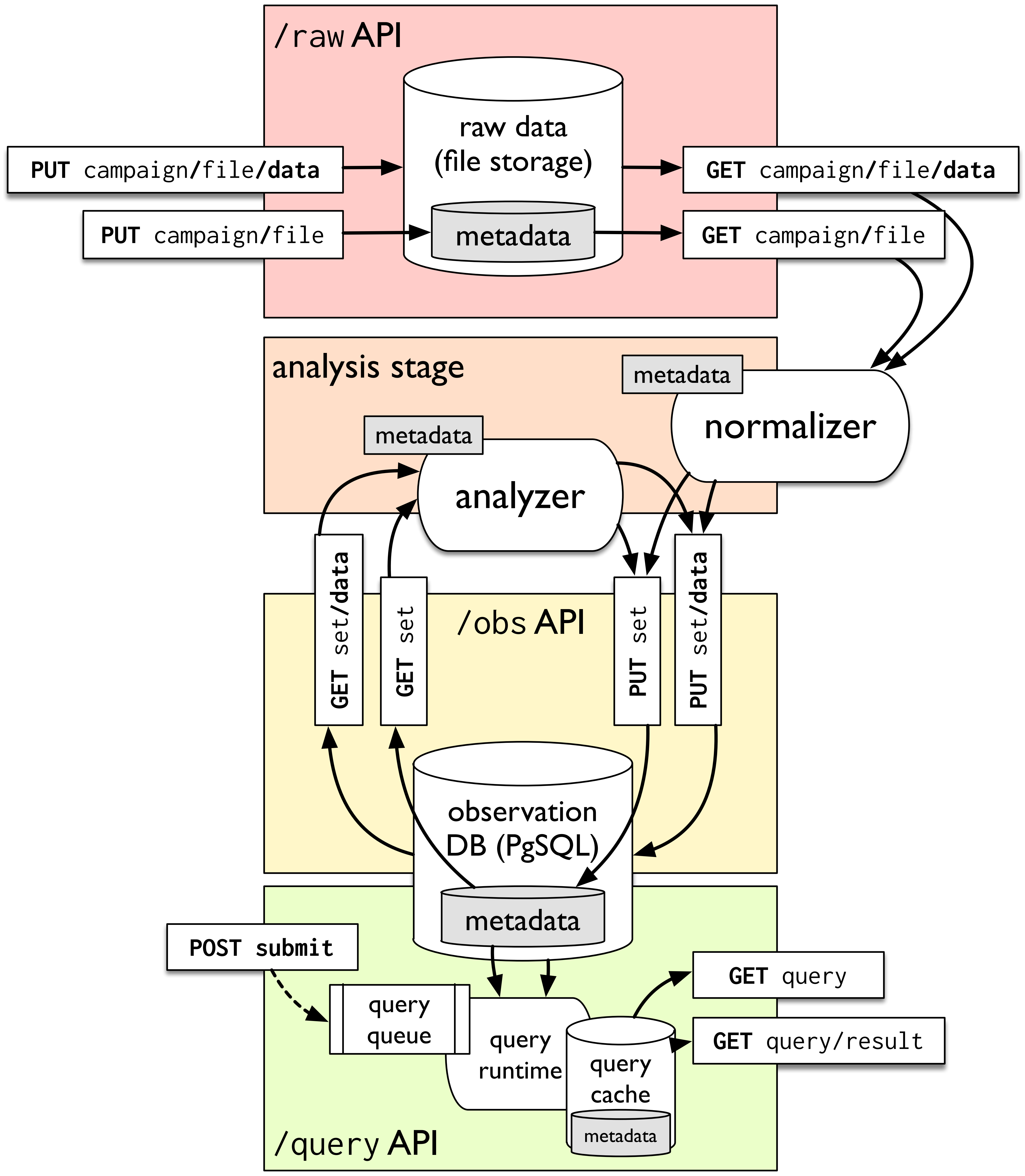
Metadata Flow

- PTO's design is **metadata-first:**
  - All additions consist of a metadata phase, then a data phase.
  - Normalizers and analyzers are controlled indirectly by raw and observation set metadata.
  - Data is immutable, but metadata can be updated.
- Arbitrary metadata: we don't know as well as the users what will be useful in the future.



PTO RESTful API

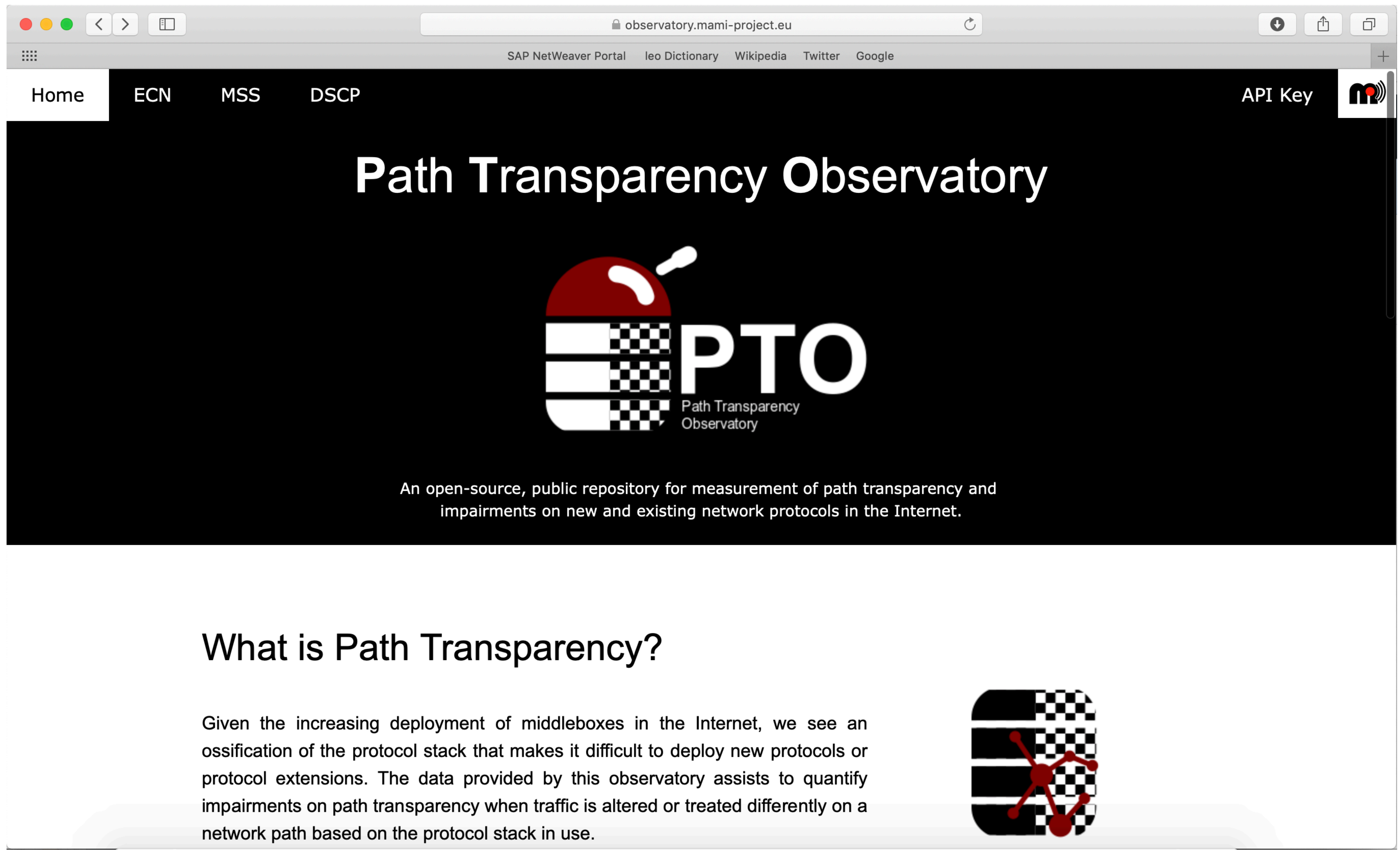
- ***Raw data*** is stored as *files* in original output format of each tool/source
  - "metadata first"
- ***Analysis*** stage derives ***observations*** in a common schema
  - organized into sets sharing provenance and other metadata
- Flexible ***query*** engine to access observation storage
  - allows query results to be cached and annotated for publication



Querying Observations

- Code: https://github.com/mami-project/pto3-go
- Doc: https://github.com/mami-project/pto3-go/tree/master/doc
- Web frontend: https://observatory.mami-project.eu
- e.g. `GET https://observatory.mami-project.eu/query/submit?time_start=2014-01-01T00%3A00%3A00Z&time_end=2020-12-31T23%3A59%3A59Z&aspect=ecn.connectivity`

Parameter	Semantics	Meaning
`time_start`	temporal	Select observations starting at or after the given start time
`time_end`	temporal	Select observations ending at or before the given end time
`set`	select	Select observations with in the given set ID
`on_path`	select	Select observations with the given element in the path
`source`	select	Select observations with the given element at the start of the path
`target`	select	Select observations with the given element at the end of the path
`condition`	select	Select observations with the given condition, with wildcards
`group`	group	Group observations and return counts by group
`option`	options	Specify a query option

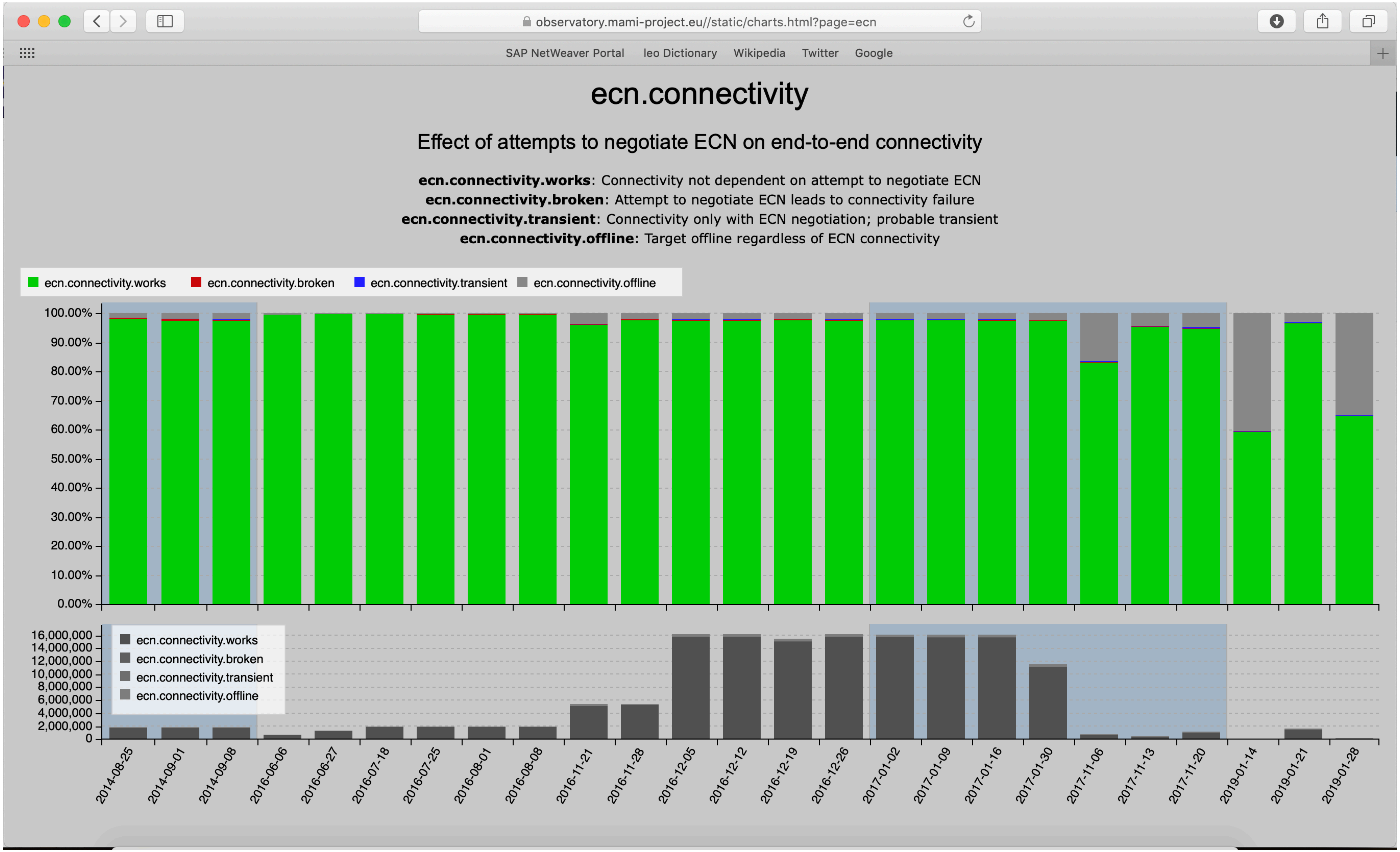


Path Transparency Observatory

An open-source, public repository for measurement of path transparency and impairments on new and existing network protocols in the Internet.

What is Path Transparency?

Given the increasing deployment of middleboxes in the Internet, we see an ossification of the protocol stack that makes it difficult to deploy new protocols or protocol extensions. The data provided by this observatory assists to quantify impairments on path transparency when traffic is altered or treated differently on a network path based on the protocol stack in use.



observatory.mami-project.eu//static/charts.html?page=ecn

SAP NetWeaver Portal   leo Dictionary   Wikipedia   Twitter   Google

ecn.connectivity

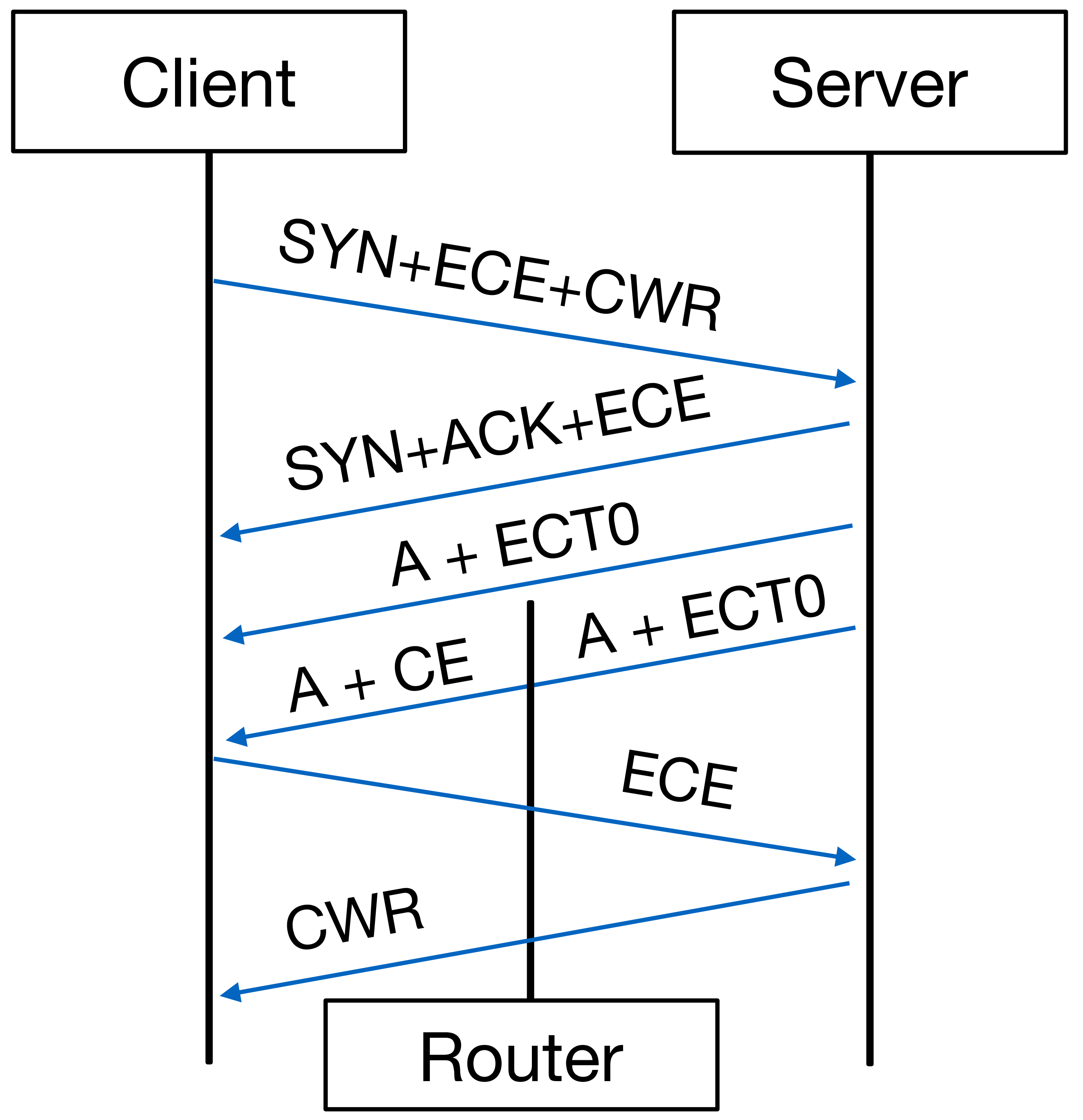
Effect of attempts to negotiate ECN on end-to-end connectivity

ecn.connectivity.works: Connectivity not dependent on attempt to negotiate ECN
ecn.connectivity.broken: Attempt to negotiate ECN leads to connectivity failure
ecn.connectivity.transient: Connectivity only with ECN negotiation; probable transient
ecn.connectivity.offline: Target offline regardless of ECN connectivity

Explicit Congestion Notification (ECN) (a specific "new" feature we'll measure today)

TCP extension allowing routers to signal congestion using bits in the IP header.

Connect **with and without ECN**

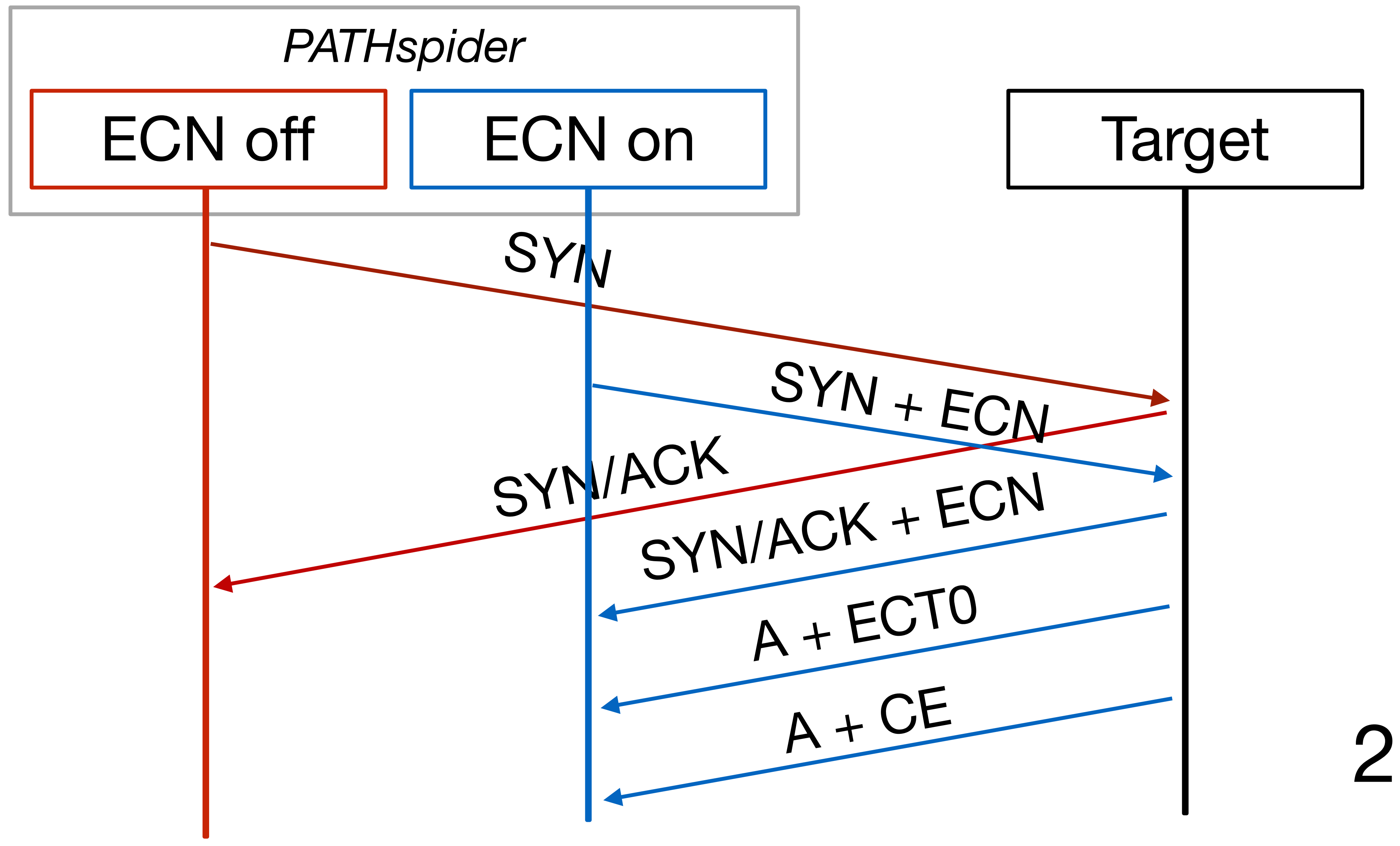


- Client attempts to negotiate ECN with TCP flags
- Server acknowledges negotiation and begins signaling ECN Capable (ECT0)
- Routers on path can mutate ECT0 to CE to note congestion (instead of tail dropping)
- Client echoes CE, server acks reduction of congestion window

The History of ECN

- Defined in RFC 3168 in 2001
    - Reused two bits from old IP TOS byte for ECT/CE signaling in the IP header
- Deployment was slow due to early problems
    - ECN-marked traffic caused some routers to reboot

Example study: ECN PATHspider Plugin

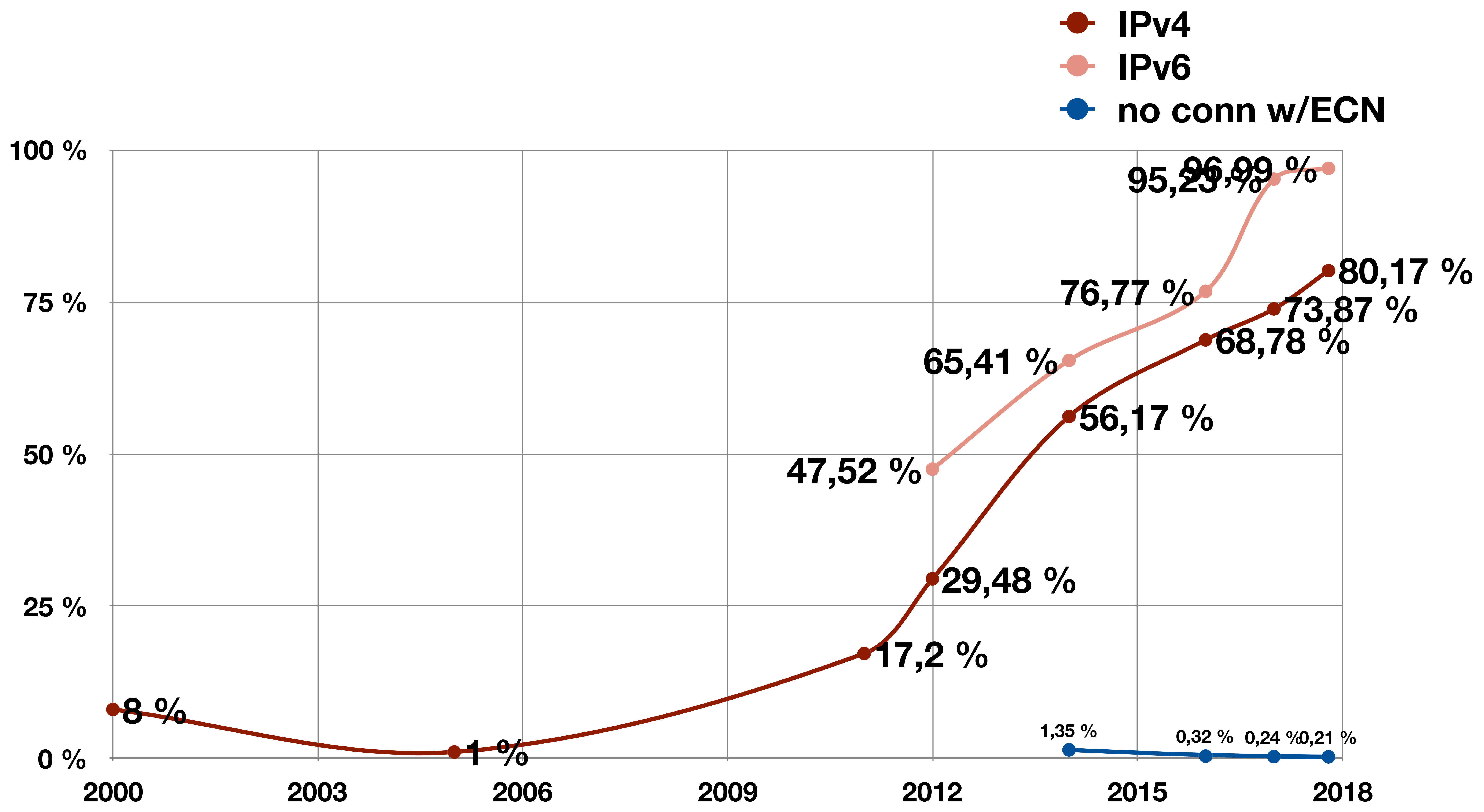


1. `ecn.connectivity.`*`status`*
   `works`: off + on OK
   `broken`: off OK, on fails
   `transient`: on OK, off fails
   `offline`: no connection

2. `ecn.`*`negotiation`*`.succeeded/failed`

3. `ecn.ipmark.ECT`*`1`*`/ECT0/CE.seen/not_seen`
   CE = Congestion Experienced; ECT = ECN Capable Transport

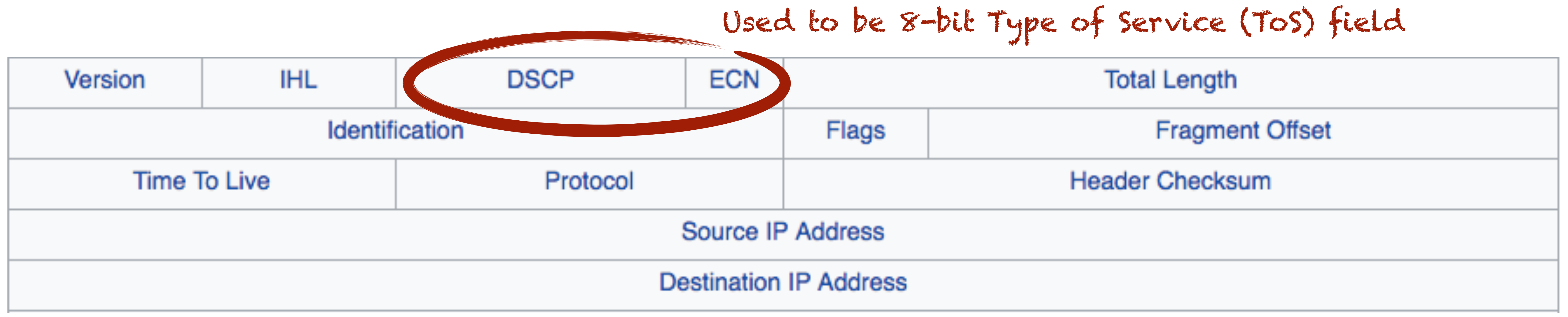
ECN support on webservers (Alexa 1Mio)



Enhanced ECN PATHspider plugin

- **Goal:** detect mangling of the IP ECN codepoint as well as DiffServ codepoint (set to 46=Expedited Forwarding)

- 4 TCP connections
  - Baseline: TCP SYN without ECN negotiation attempt
  - TCP SYN with ECN negotiation + no ECN IP codepoit (Not-ECT)
  - TCP SYN with or without ECN negotiation + ECT(1)
  - TCP SYN with or without ECN negotiation + CE


Used to be 8-bit Type of Service (ToS) field

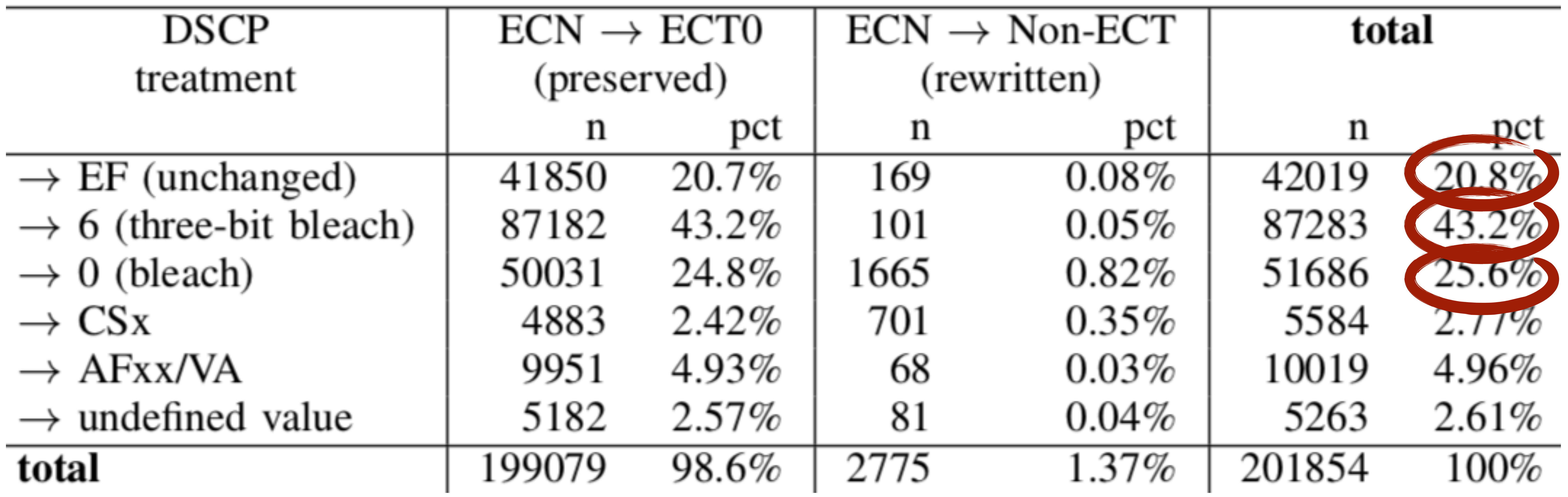
Version	IHL	DSCP	ECN	Total Length	
Identification				Flags	Fragment Offset
Time To Live		Protocol		Header Checksum	
Source IP Address					
Destination IP Address					

ECN negotiation with IP ECN codepoint

- 69.35% only negotiated ECN when the ECN IP codepoint was set to zeros (non-ECT) but not if ETC0 or CE was set
- Only 12.79% of the hosts negotiated ECN no matter what codepoint was set
- 26 hosts negotiated ECN when ECT0 was set but not when CE was set

DSCP and ECN IP Codepoint Manipulation

without ECN negotiationn but ECT(0) (for 201,854 hosts)

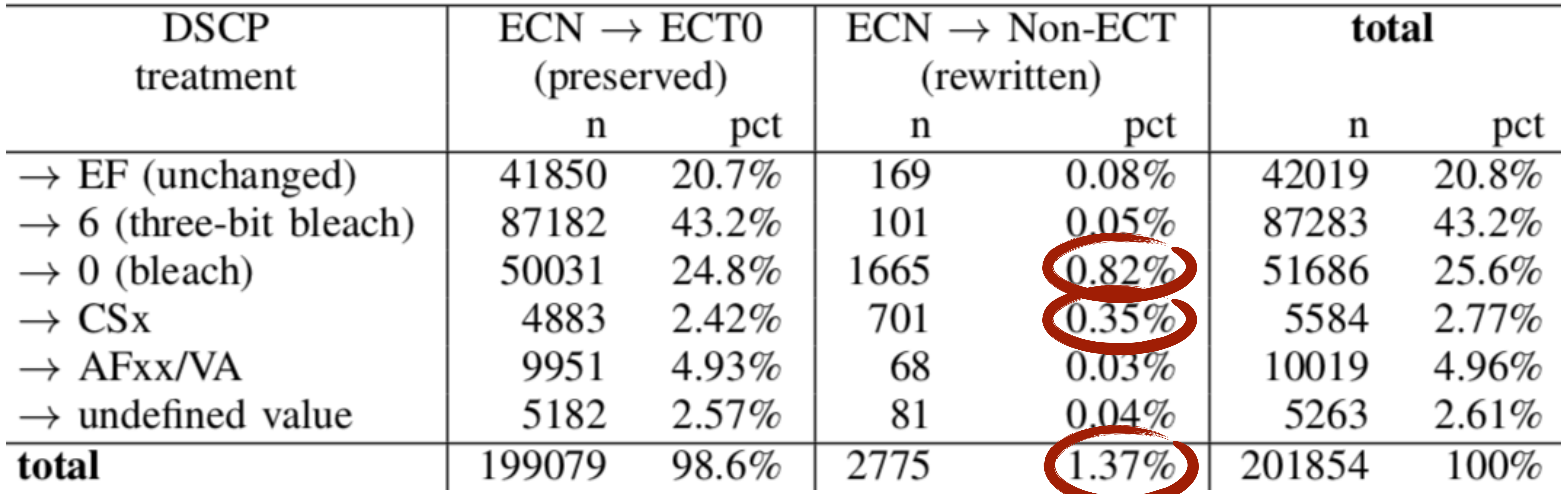


DSCP treatment	ECN → ECT0 (preserved)		ECN → Non-ECT (rewritten)		**total**	
	n	pct	n	pct	n	pct
→ EF (unchanged)	41850	20.7%	169	0.08%	42019	20.8%
→ 6 (three-bit bleach)	87182	43.2%	101	0.05%	87283	43.2%
→ 0 (bleach)	50031	24.8%	1665	0.82%	51686	25.6%
→ CSx	4883	2.42%	701	0.35%	5584	2.77%
→ AFxx/VA	9951	4.93%	68	0.03%	10019	4.96%
→ undefined value	5182	2.57%	81	0.04%	5263	2.61%
total	199079	98.6%	2775	1.37%	201854	100%

- Side note: 3,252 hosts reflected ECT(0) in the SYN/ACK even though ECN was not requested in the SYN
- Four of five paths see some DSCP manipulation
- Bleaching most common (incl. 3-bit ToS IP Precedence bits)

DSCP and ECN IP Codepoint Manipulation

without ECN negotiation but with ECT(1) (for 201,854 hosts)



DSCP treatment	ECN → ECT0 (preserved)		ECN → Non-ECT (rewritten)		**total**	
	n	pct	n	pct	n	pct
→ EF (unchanged)	41850	20.7%	169	0.08%	42019	20.8%
→ 6 (three-bit bleach)	87182	43.2%	101	0.05%	87283	43.2%
→ 0 (bleach)	50031	24.8%	1665	0.82%	51686	25.6%
→ CSx	4883	2.42%	701	0.35%	5584	2.77%
→ AFxx/VA	9951	4.93%	68	0.03%	10019	4.96%
→ undefined value	5182	2.57%	81	0.04%	5263	2.61%
total	199079	98.6%	2775	1.37%	201854	100%

- For 2,775 (1.37%) hosts, the ECT0 codepoint was erased before received at server
  - about 50% of the codepoint removal in the last hop
  - more than 90% of the cases in the last 40% of the path
- The majority of ECN-manipulating paths also bleach the DSCP codepoint
- ECN codepoint is set to 0 while the DSCP codepoint is set to a CS value
  - indicates treatment according to the old ToS definition :-(

Summary

- Measure **Internet Path Transparency** to detect and quantify network impairments as input for protocol design
- **PATHspider** - A framework for testing Path Transparency
- **PTO** - A centralized storage and analysis architecture
- ***Example ECN:*** Majority of on-path interference with the ECN IP codepoint is linked to older interpretation of the ToS byte
    - While ToS bleaching was observed on the whole network path (border routers), active ECN IP rewriting is more commonly performed at edge networks (based on additional traceroute measurements)

A few simple principles go a long way.

- Using common measurement frameworks and pulling data together into a ***simple schema*** with a small set of metrics/semantics makes comparability possible.

- Measurement ***metadata*** is important: it gives you a way to express what you did and how in a way that you (and others!) can use later.
  - There's a tradeoff between ad-hoc (ease of writing) and standard-structured (ease of use) schemas here.
  - The details of what *exactly* a tool does, how that tool was configured, and the context in which a measurement was taken are important.

- ***Provenance*** is also important: knowing where every bit of data and analysis came from allows repeatability
  - Doing this all the way down to commit references / software repository release number for measurement and analysis tools makes reproducibility easier.

Overview Part II

- Hands-on work with PATHspider
  - Let’s start some measurements!
  - Target Lists using Hellfire
  - PATHspider Plugins
  - Use of scapy for packet forging
  - Observations and Analysis

Run a measurement!

- Go into the PATHspider git folder
- Start Vagrant

```
$ vagrant up

$ vagrant ssh
```

- Run a measurement

```
$ sudo pspdr measure ecn < /vagrant/input/tma-tutorial.ndjson.<XX> > results.ndjson &
```

Note: Usually you need to specify n interface with `-i` however this is a special version for the tutorial (using a university VPN tunnel in order to avoid problems with the local network)

PATHspider 2.0

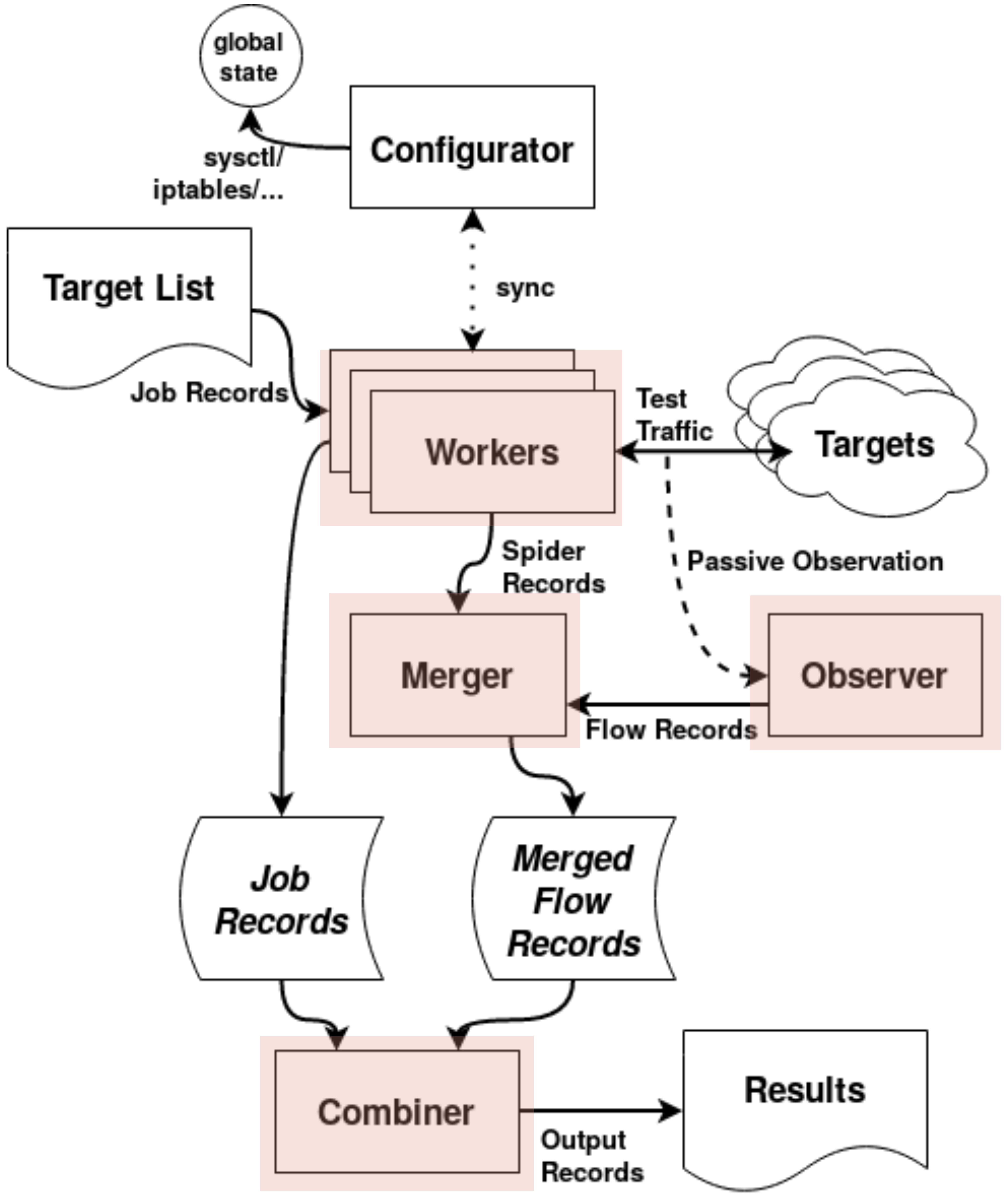
- Webpage: https://pathspider.net
- GitHub: https://github.com/mami-project/pathspider/
- Documentation: https://pathspider.readthedocs.io/en/stable/

- PATHspider is packaged for Debian and packages are made available for the testing and stable-backports distributions.
- To install PATHspider, run:

```
$ sudo apt install pathspider
```

PATHspider *2.0*

- **Workers** open the connection and send test data for all target on target list
- **Observer** passively monitors all out-going and in-coming packets
- **Merger** appends record information from each worker to the passively observed flow records
- **Combiner** analyses the results of all connections attempts belonging to one test and generates *observations*



Using PATHspider

```
$ pspdr
usage: pspdr [-h] [--verbose] COMMAND ...

PATHspider will spider the paths.

optional arguments:
  -h, --help  show this help message and exit
  --verbose   Enable verbose logging

Commands:
    filter    Pre-process a target list
    measure   Perform a PATHspider measurement
    metadata  Create PTOv3 metadata files from results
    upload    Uploads data to PTO Creates metadata if not provided
    observe   Passively observe network traffic
    test      Run the built in test suite

Spider safely!
```

Using PATHspider plugins to measure

```
$ pspdr measure --help
usage: pspdr measure [-h] [-i INTERFACE] [-w WORKERS] [--input INPUTFILE]
                     [--csv-input] [--output OUTPUTFILE] [--output-flows]
                     PLUGIN ...

optional arguments:
  -h, --help            show this help message and exit
  -i INTERFACE, --interface INTERFACE
                        The interface to use for the observer. (Default: tun0)
  -w WORKERS, --workers WORKERS
                        Number of workers to use. (Default: 20)
  --input INPUTFILE     A file containing a list of PATHspider jobs. Defaults
                        to standard input.
  --csv-input           Indicate CSV format.
  --output OUTPUTFILE   The file to output results data to. Defaults to
                        standard output.
  --output-flows        Include flow results in output.

Plugins:
  The following plugins are available for use:

    tfo                 TCP Fast Open
    mss                 TCP Maximum Segment Size
    tcpopt              TCP Options (Timestamp, Windows Scaling, SACK)
    ecn                 Explicit Congestion Notification
    evilbit             Evil bit connectivity testing
    h2                  HTTP/2
    dnsresolv           Simple Input List DNS Resolver
    dscp                Differentiated Services Codepoints
    udpzero             UDP Zero Checksum

Spider safely!
```

Using PATHspider as Passive Observer

- Observer modules used to be part of plugins in PATHspider 1.0, but in 2.0 they are independent and so can be reused across multiple plugins:
  - BasicChain, DNSChain, DSCPChain, ECNChain, EvilChain, ICMPChain, TCPChain, TFOChain

```
$ sudo pspdr observe --list-chains
```

- These can also be used together, limiting each chain to just a single layer and letting the combiner produce conditions
- Chains can produce information to be consumed by other chains later in the list
- These can be used independently of a PATHspider measurement, e.g.:

```
$ sudo pspdr observe tcp ecn
```

Target Lists using Hellfire

- **Hellfire** is a parallelised DNS resolver to generate input lists for PATHspider
  - But can also be use be used other applications!
  - Written in Go and available on GitHub: https://github.com/mami-project/hellfire
  - Various input sources supported:
    - Alexa Top 1 Million Global Sites
    - Cisco Umbrella 1 Million
    - Citizen Lab Test Lists
    - OpenDNS Public Domain Lists
    - Comma-Seperated Values Files
    - Plain Text Domain Lists

Hellfire Usage

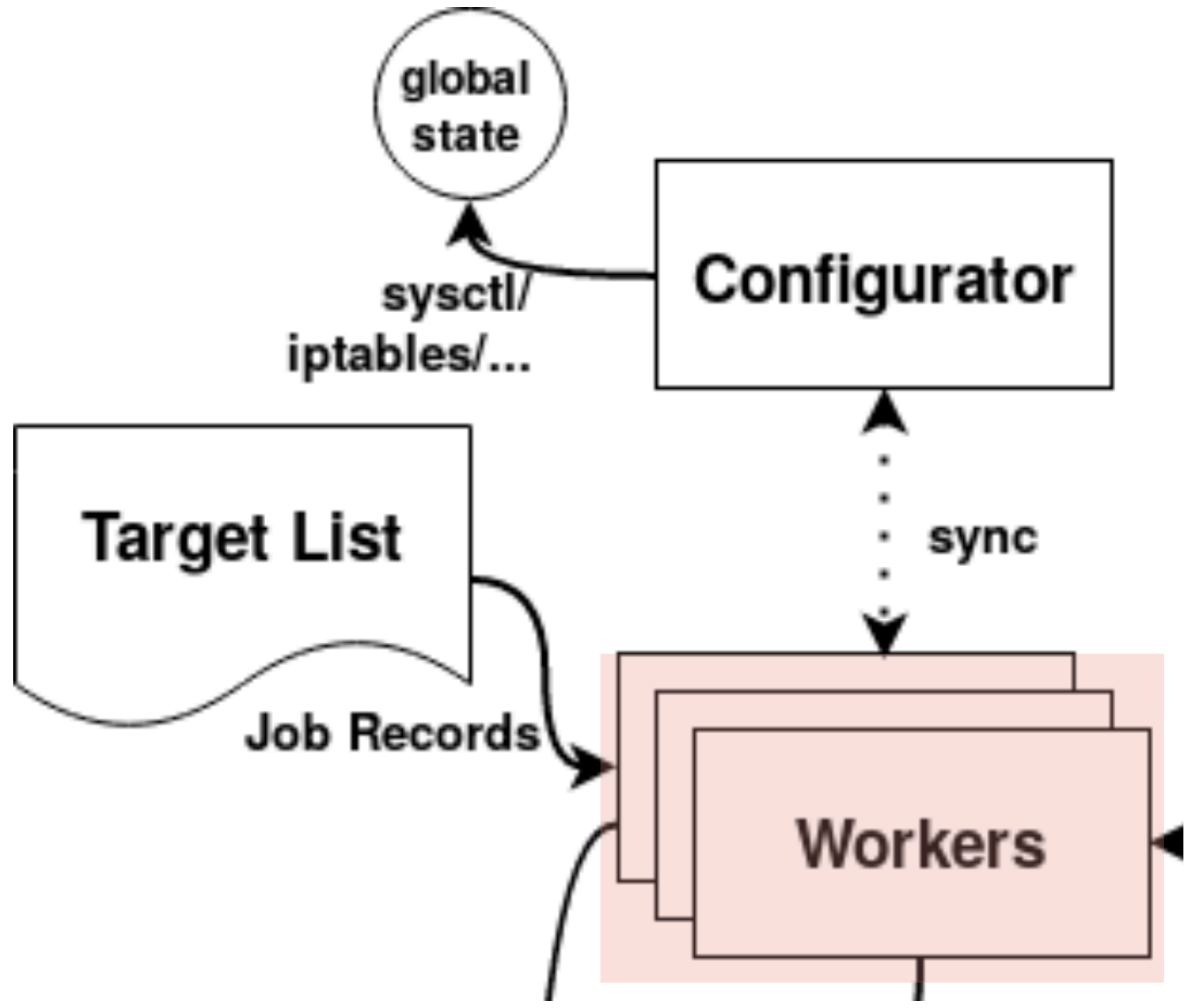
```
$ hellfire
Usage :
   hellfire --topsites [--file=<filename>] [--output=<individual|array|oneeach>] [--type=<
      host | ns |mx>] [--canid=<canid address >]
   hellfire --cisco [--file=<filename>] [--output=<individual|array|oneeach>] [--type=<
      host | ns |mx>] [--canid=<canid address >]
   hellfire --citizenlab [--country=<cc>|--file=<filename>] [--output=<individual|array|
      oneeach >] [--type=<host | ns |mx>] [--canid=<canid address >]
   hellfire --opendns [--list=<name>|--file=<filename>] [--output=<individual|array|
      oneeach >] [--type=<host | ns |mx>] [--canid=<canid address >]
   hellfire --csv --file=<filename> [--output=<individual|array|oneeach>] [--type=<host|ns
      |mx>] [--canid=<canid address >]
   hellfire --txt --file=<filename> [--output=<individual|array|oneeach>] [--type=<host|ns
      |mx>] [--canid=<canid address >]
```

```
$ hellfire --cisco
```

PATHspider Plugins

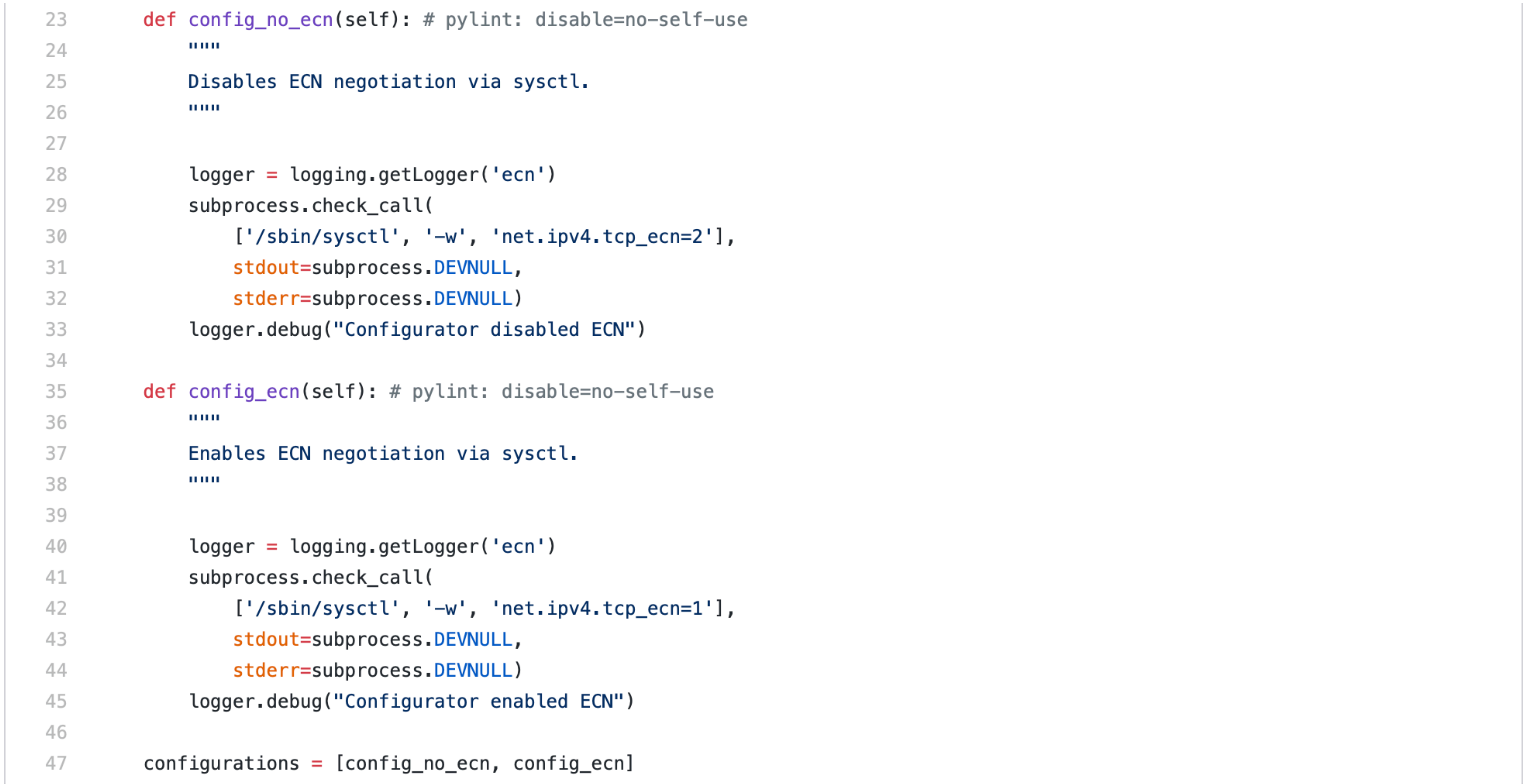
- Synchronised (traditional ecnspider)
    - ECN, DSCP
- Desynchronised (traditional ecnspider, no configurator)
    - TFO, H2, TLS NPN/ALPN
- Forge (new in PATHspider 2.0.0)
    - Evil Bit, UDP Zero Checksum, UDP Options
- Single (new in PATHspider 2.0.0 and fast)
    - Various TCP Options

Synchronized Plugin



- SynchronizedSpider plugins use built-in connection methods along with global system configuration to change the behaviour of the connections
- One **function** should be written for each **configuration**
  - Configuration functions may make calls to `sysctl` or `iptables` to make ***global*** changes to the way that traffic is generated
  - PATHspider will ensure that the configurations are set before the corresponding traffic is generated
  - However, plugin authors must ensure that any configuration is reset by the next configuration function if that is required!

Example ECN Plugin



```
    def config_no_ecn(self): # pylint: disable=no-self-use
        """
        Disables ECN negotiation via sysctl.
        """

        logger = logging.getLogger('ecn')
        subprocess.check_call(
            ['/sbin/sysctl', '-w', 'net.ipv4.tcp_ecn=2'],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL)
        logger.debug("Configurator disabled ECN")

    def config_ecn(self): # pylint: disable=no-self-use
        """
        Enables ECN negotiation via sysctl.
        """

        logger = logging.getLogger('ecn')
        subprocess.check_call(
            ['/sbin/sysctl', '-w', 'net.ipv4.tcp_ecn=1'],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL)
        logger.debug("Configurator enabled ECN")

    configurations = [config_no_ecn, config_ecn]
```

Desynchronized Plugin

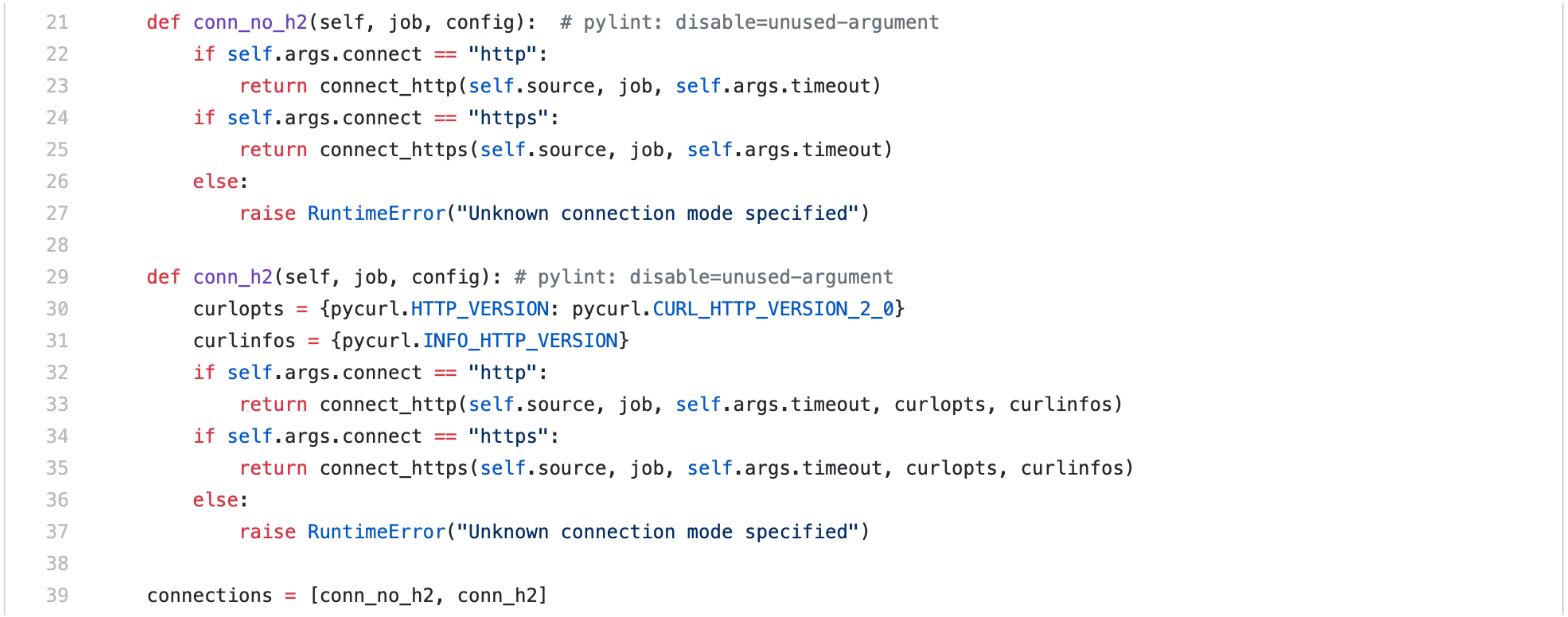
- For DesynchronizedSpider plugins, there is no global state synchronisation.
- One **function** should be written for each **connection** to be made:
    - Connection functions will modify the ***connection logic directly***
    - Connection helper (or custom connection logic) should be used to generate traffic to target and to get a reply from the target
    - Usually at least two functions need to be provided - a baseline followed by an experimental connection.
- DesynchronizedSpider can be more efficient than a SynchronizedSpider!

.

Connection Helpers

- Instead of writing client code, use the code that already exists In the **`pathspider.helpers`** module:
  - DNS (dnslib)
  - HTTP/HTTPS (pycURL)
  - TCP (Python socket)
- For synchronised plugins, just use the helper
- For desycnhronised plugins, the helpers are customisable, e.g. cURL helpers accept arbitrary CURLOPTs

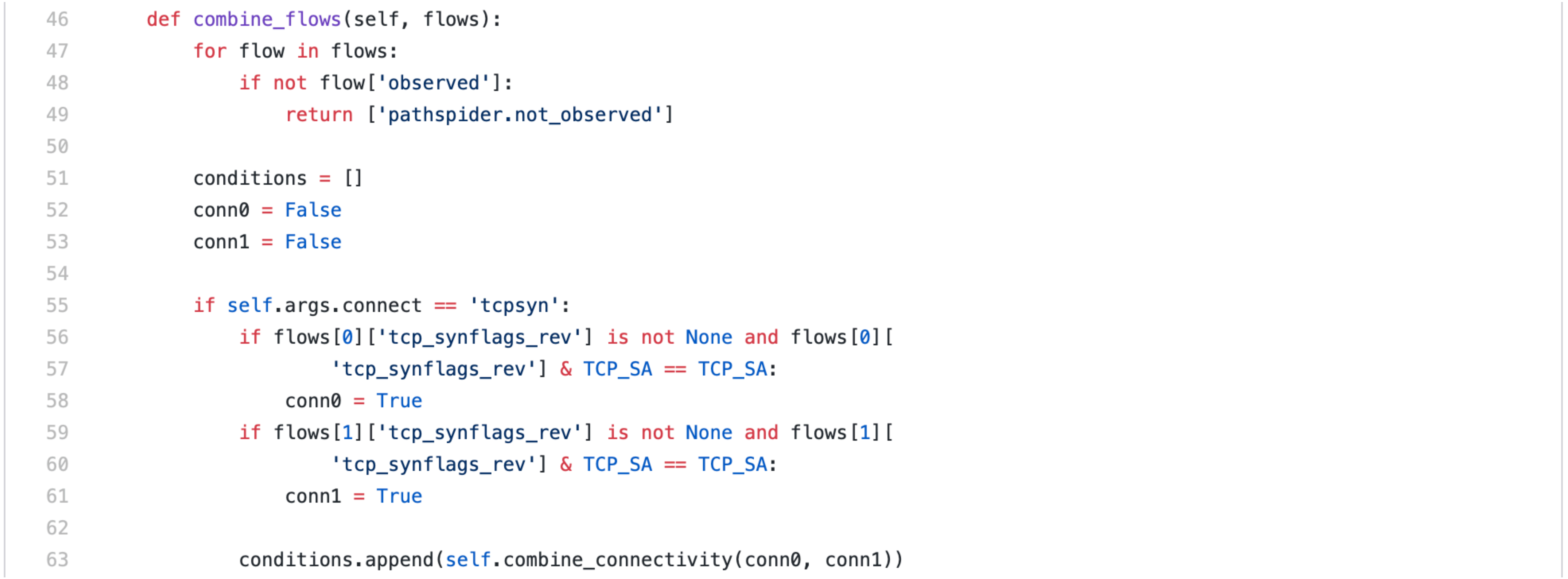
Example H2 plugin



```python
def conn_no_h2(self, job, config):  # pylint: disable=unused-argument
    if self.args.connect == "http":
        return connect_http(self.source, job, self.args.timeout)
    if self.args.connect == "https":
        return connect_https(self.source, job, self.args.timeout)
    else:
        raise RuntimeError("Unknown connection mode specified")

def conn_h2(self, job, config): # pylint: disable=unused-argument
    curlopts = {pycurl.HTTP_VERSION: pycurl.CURL_HTTP_VERSION_2_0}
    curlinfos = {pycurl.INFO_HTTP_VERSION}
    if self.args.connect == "http":
        return connect_http(self.source, job, self.args.timeout, curlopts, curlinfos)
    if self.args.connect == "https":
        return connect_https(self.source, job, self.args.timeout, curlopts, curlinfos)
    else:
        raise RuntimeError("Unknown connection mode specified")

connections = [conn_no_h2, conn_h2]
```

Single Plugin

- SingleSpider uses the built-in connection helpers to make a single connection to the target (which is observed by Observer chains)
- Plugin only requires a **`combine_flows()`** function to generate conditions:



```
    def combine_flows(self, flows):
        for flow in flows:
            if not flow['observed']:
                return ['pathspider.not_observed']

        conditions = []
        conn0 = False
        conn1 = False

        if self.args.connect == 'tcpsyn':
            if flows[0]['tcp_synflags_rev'] is not None and flows[0][
                    'tcp_synflags_rev'] & TCP_SA == TCP_SA:
                conn0 = True
            if flows[1]['tcp_synflags_rev'] is not None and flows[1][
                    'tcp_synflags_rev'] & TCP_SA == TCP_SA:
                conn1 = True

            conditions.append(self.combine_connectivity(conn0, conn1))
```

Forge Plugin

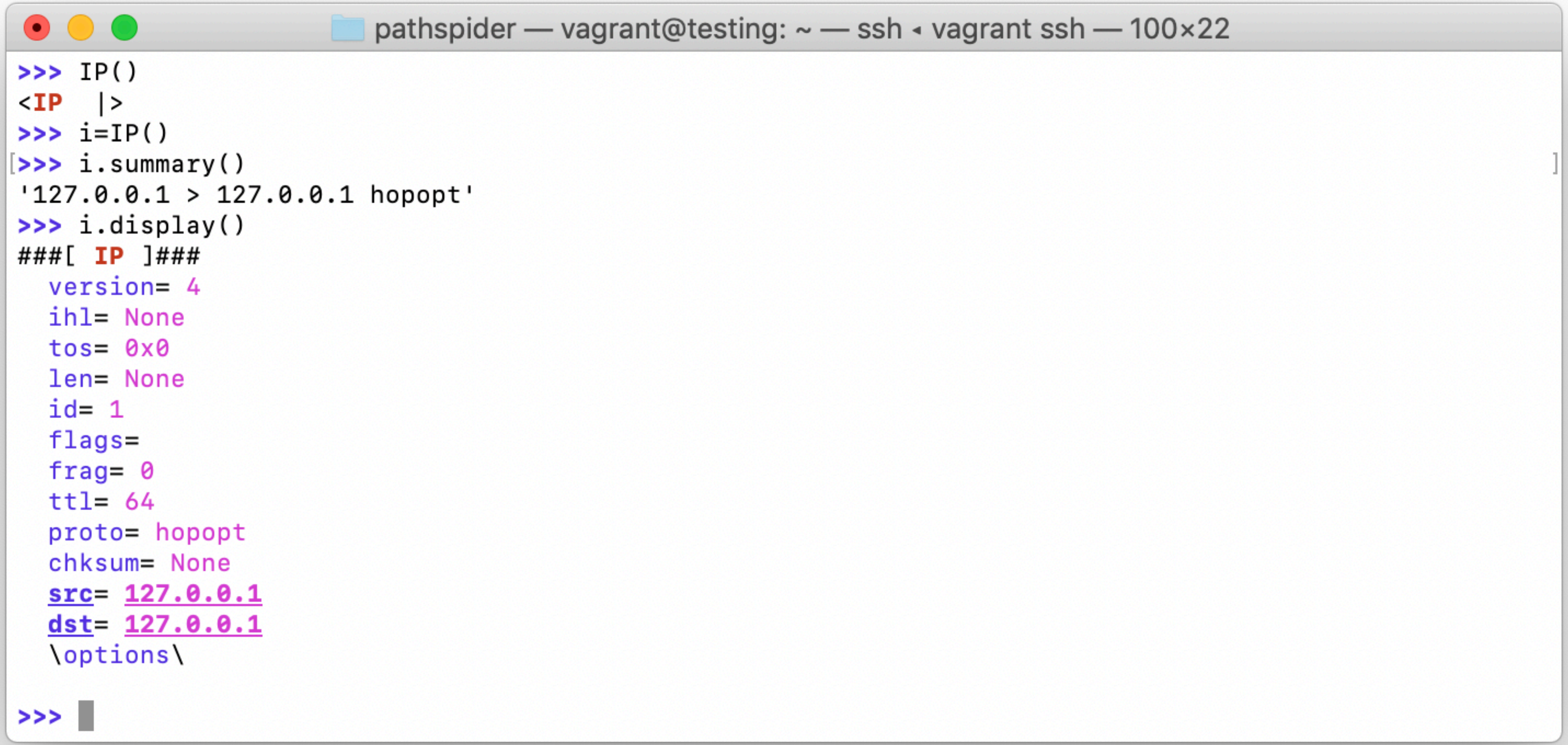
- ForgeSpider plugins use Scapy to send forged packets to targets
- The heart of a ForgeSpider is the **`forge()`** function
    - takes two arguments: the ***job*** containing the target information and the ***sequence number***
    - This function will be called the number of times set in the packets metadata variable and seq will be set to the number of times the function has been called for this job
- PATHspider uses the ***Scapy*** library for Python for packet forging
    - This is the most flexible method of creating new measurement plugins for PATHspider!

Scapy: a toolkit for building anything else you might need

- Packet manipulation tool to create or decode packets layer by layer, header by header, byte by byte.
- While you can specify raw bytes, Scapy provides a number of useful classes for common protocols, which makes things a lot easier
  - Solidly in "hack up a quick little script" territory →
    be careful with metadata/provenance for these tools
- Scapy must be launched with sudo as we will need to use “raw” sockets to emit forged packets:

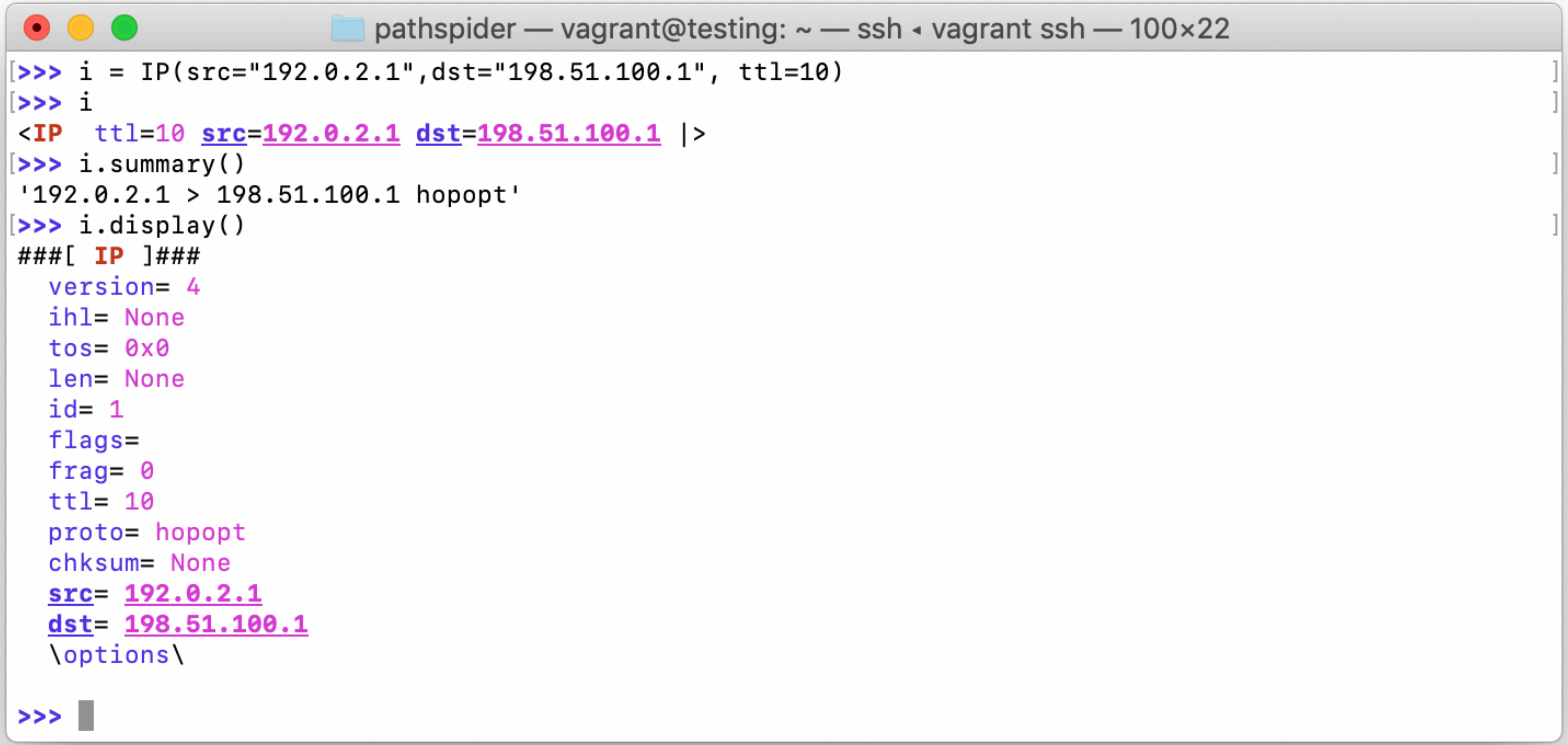
```
$ sudo scapy
 >>>
```

Make a packet: IPv4 header - create and dissect


pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×22

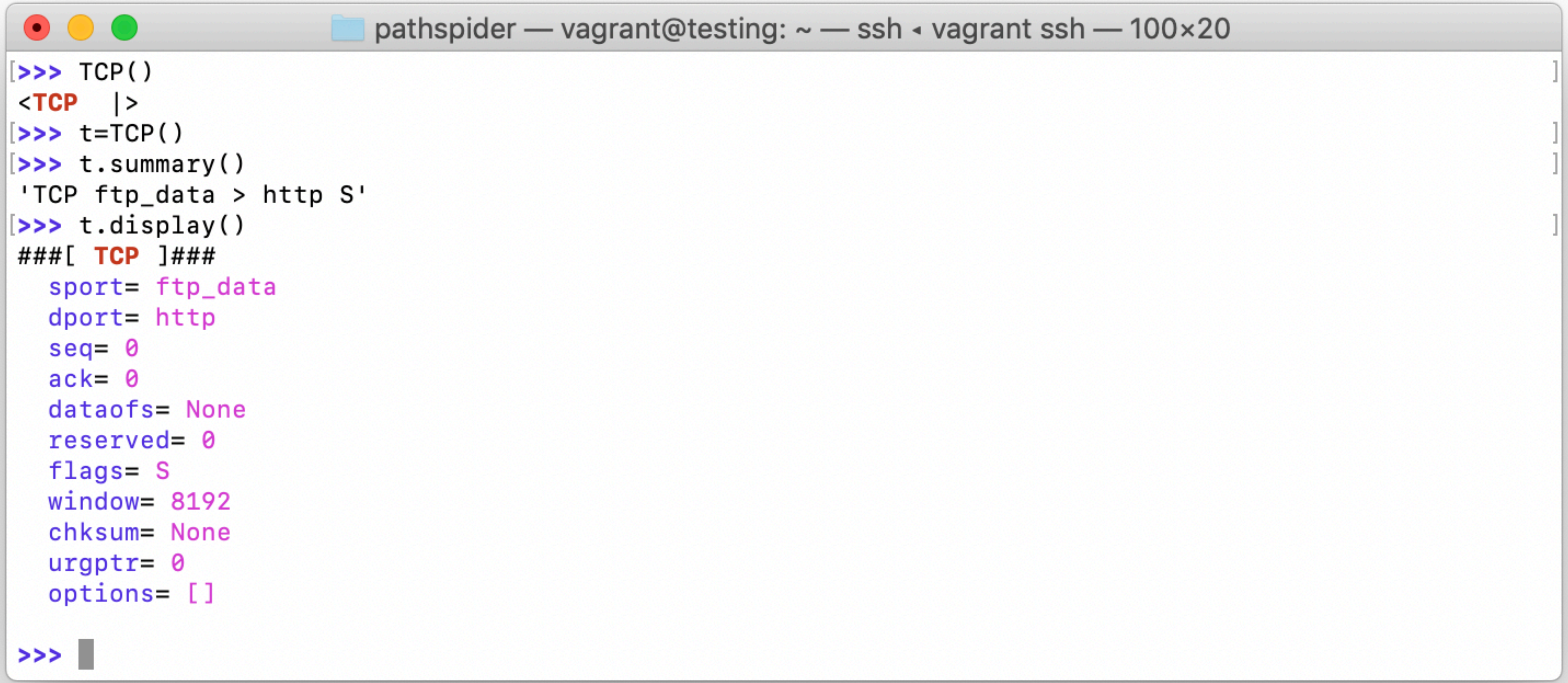
```
>>> IP()
<IP  |>
>>> i=IP()
>>> i.summary()
'127.0.0.1 > 127.0.0.1 hopopt'
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= hopopt
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\

>>>
```

Make a packet: IPv4 header - customize


pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×22

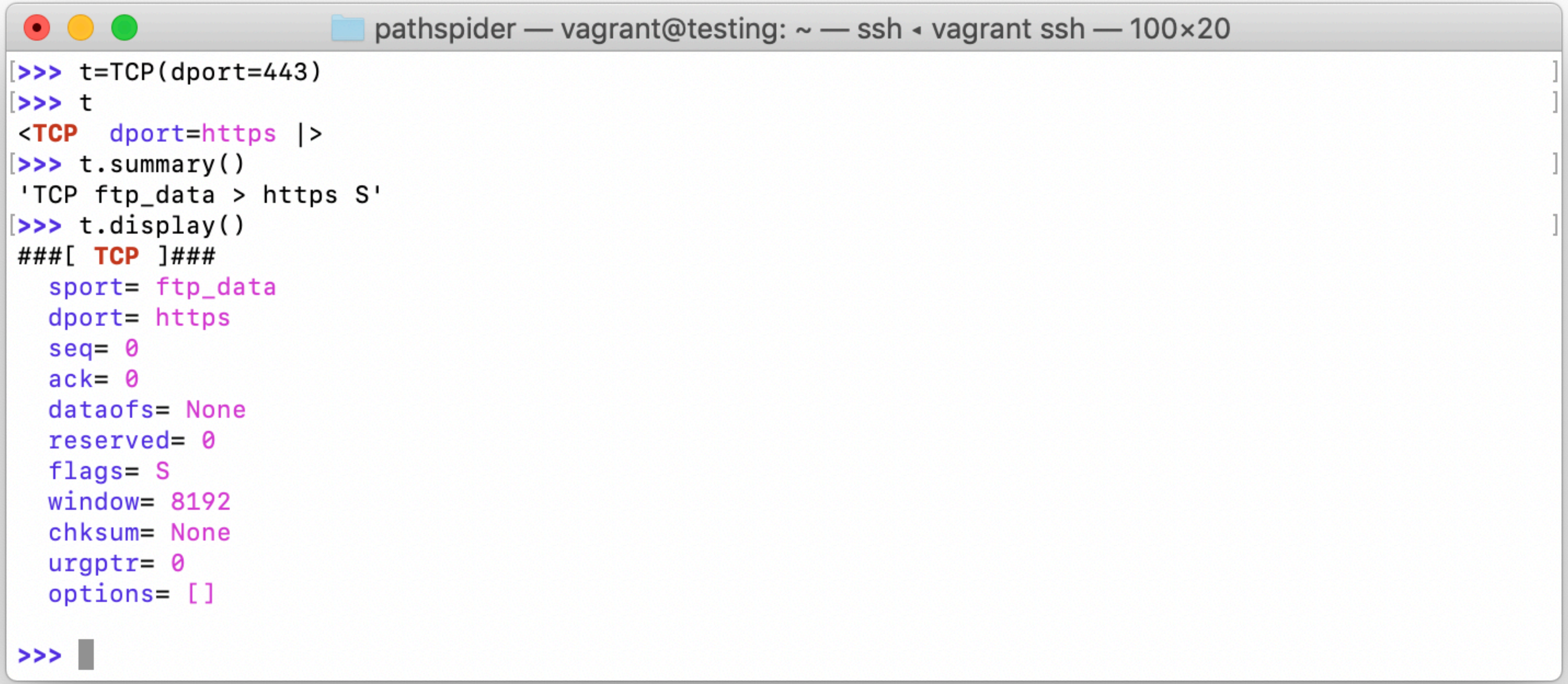
```
>>> i = IP(src="192.0.2.1",dst="198.51.100.1", ttl=10)
>>> i
<IP  ttl=10 src=192.0.2.1 dst=198.51.100.1 |>
>>> i.summary()
'192.0.2.1 > 198.51.100.1 hopopt'
>>> i.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 10
  proto= hopopt
  chksum= None
  src= 192.0.2.1
  dst= 198.51.100.1
  \options\

>>>
```

Make a packet: TCP header - create and dissect


pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×20

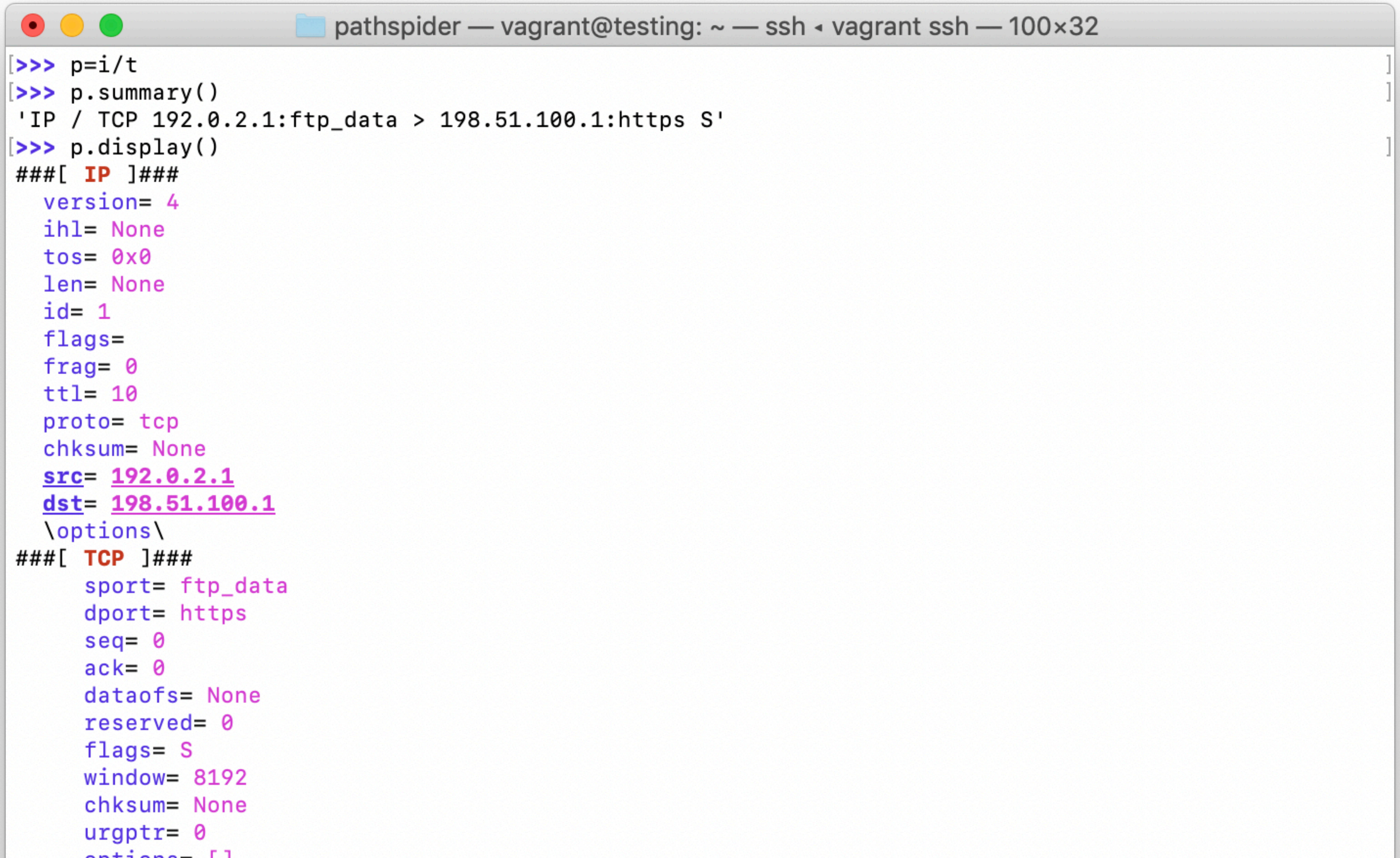
```
>>> TCP()
<TCP  |>
>>> t=TCP()
>>> t.summary()
'TCP ftp_data > http S'
>>> t.display()
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= []

>>> 
```

Make a packet: TCP header - customise


pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×20

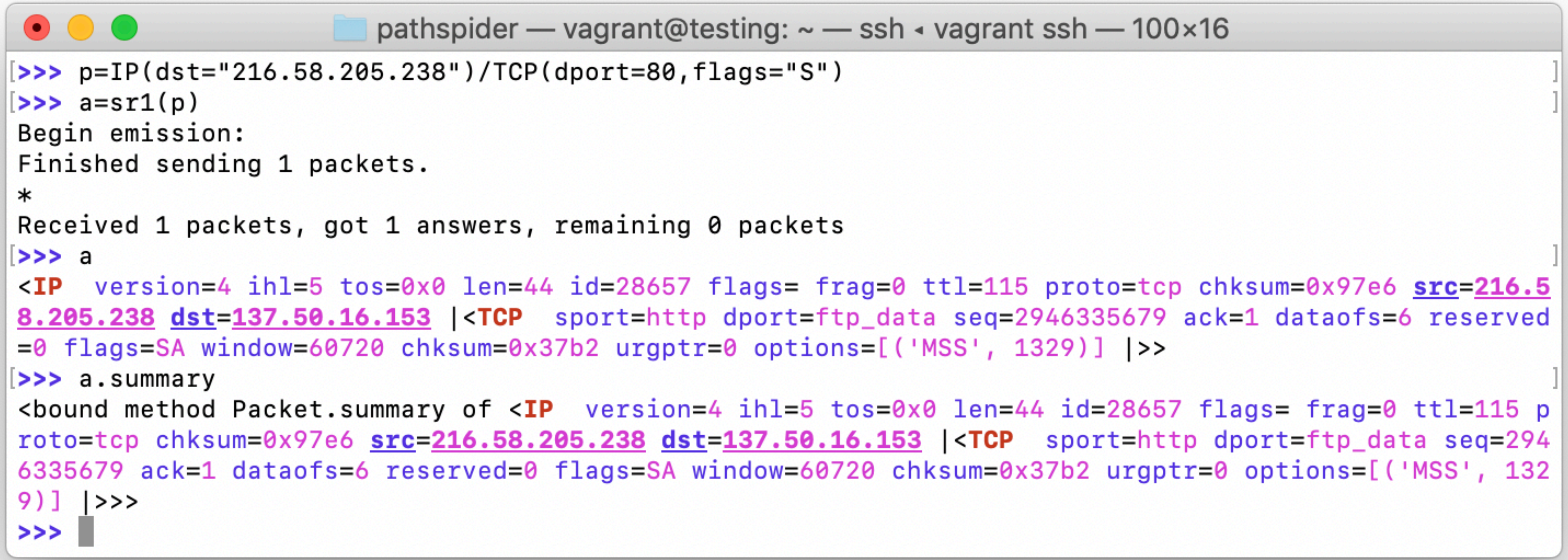
```
>>> t=TCP(dport=443)
>>> t
<TCP  dport=https |>
>>> t.summary()
'TCP ftp_data > https S'
>>> t.display()
###[ TCP ]###
  sport= ftp_data
  dport= https
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgptr= 0
  options= []

>>> 
```

Make a packet: Sticking the Pieces Together


pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×32

```
[>>> p=i/t
[>>> p.summary()
'IP / TCP 192.0.2.1:ftp_data > 198.51.100.1:https S'
[>>> p.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 10
  proto= tcp
  chksum= None
  src= 192.0.2.1
  dst= 198.51.100.1
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= https
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= []
```

Send a packet



pathspider — vagrant@testing: ~ — ssh ◂ vagrant ssh — 100×16

```
>>> p=IP(dst="216.58.205.238")/TCP(dport=80,flags="S")
>>> a=sr1(p)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> a
<IP  version=4 ihl=5 tos=0x0 len=44 id=28657 flags= frag=0 ttl=115 proto=tcp chksum=0x97e6 src=216.5
8.205.238 dst=137.50.16.153 |<TCP  sport=http dport=ftp_data seq=2946335679 ack=1 dataofs=6 reserved
=0 flags=SA window=60720 chksum=0x37b2 urgptr=0 options=[('MSS', 1329)] |>>
>>> a.summary
<bound method Packet.summary of <IP  version=4 ihl=5 tos=0x0 len=44 id=28657 flags= frag=0 ttl=115 p
roto=tcp chksum=0x97e6 src=216.58.205.238 dst=137.50.16.153 |<TCP  sport=http dport=ftp_data seq=294
6335679 ack=1 dataofs=6 reserved=0 flags=SA window=60720 chksum=0x37b2 urgptr=0 options=[('MSS', 132
9)] |>>>
>>> 
```

Writing a Plugin: The Evil Bit

- To get started you will need the required directory layout for PATHspider plugins, in this case for the EvilBit plugin:

```
+-- pathspider
      +-- __init__.py
      +-- plugins
          +-- __init__.py
          +-- evilbit.py
```

- Inside both _init_.py files, you will need to add the following (and only the following):

```
from pkgutil import extend path
path = extend path( path , name )
```

- Your plugin will be written in evilbit.py and will be discovered automatically when you run PATHspider
  - Use scapy's `i.flags = ’evil’` to forge packet with evil bit set
  - Use `self.source[0]` for system IPv4 address and `self.source[1]` for Ipv6 address
  - Use `job[‘dip']` for destination IP address and `job[‘dp']` for destination port

Analysis

- E.g. use Jupyter Notebooks

```
$ cd /vagrant/
$ cp /home/results.ndjson .
$ jupyter notebook —ip 0.0.0.0
```

- Then open browser and go to

`http://localhost:8888/notebooks/analysis.ipynb`

Summary

- **PATHspider 2.0** - A generalized framework for A/B testing
  - Webpage: https://pathspider.net
  - GitHub: https://github.com/mami-project/pathspider/
  - Documentation: https://pathspider.readthedocs.io/en/stable/
- PATHspider Plugins
  - Synchronised: e.g. ECN, DSCP
  - Desynchronised (no configurator), e.g. TFO, H2, TLS NPN/ALPN
  - Forge (new in PATHspider 2.0.0): e.g. Evil Bit, UDP Zero Checksum, UDP Options
  - Single (new in PATHspider 2.0.0 and fast): e.g. various TCP Options using **Scapy**
- **Spider safely!**