WP1: PTO v3

Stephan Neuhaus, Brian Trammell

mami

measurement and architecture for a middleboxed internet

measurement | architecture | experimentation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.The opinions expressed and arguments employed reflect only the authors' view. The European Commission is not responsible for any use that may be made of that information.

PTO3 Redux

- PTO1: “Big Data!”: Spark! HDFS!! MongoDB!!1!
  - Spark: slow
  - HDFS: slow, no security
  - MongoDB: slow, no security

  } “medium data” good enough

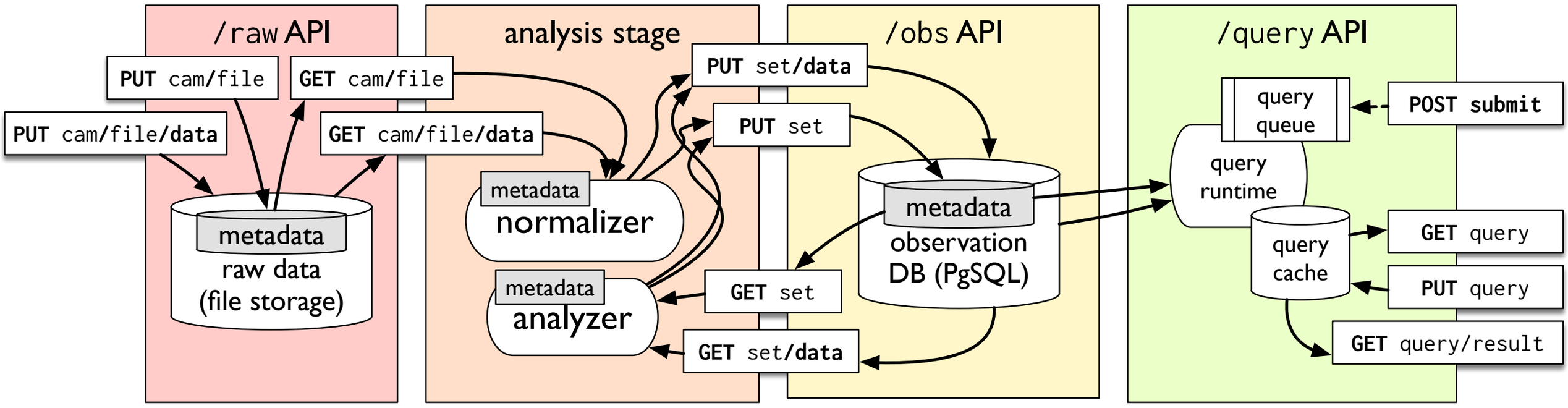
- PTO2: “Overgeneralize ALL the things!”
  - Very general data model and queries
  - IQL: beautiful and general
    - Hard to execute quickly
    - Queries can take hours

  } restrict data model restrict queries

PTO3 Design and Data Model

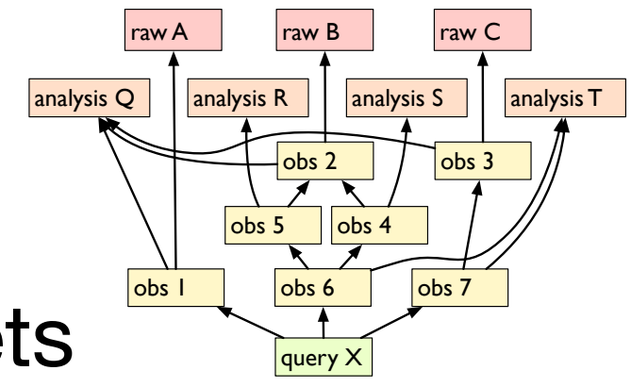
- Design centered on RESTful(ish) API, with three types of resources:
  - *raw data files* organized into *campaigns,* with associated *metadata*
  - *observation sets,* groups of *observations* sharing *provenance* and *metadata*
  - *queries*, containing *cached results, provenance* and *metadata*
- Normalizers turn data into observations.
- An observation is a path-transparency-specific assertion that...
  - On this *path*, during this *time* interval, this *condition* was measured.
- Conditions are structured names
  - <feature>.<aspect>.<subaspect>.<state> …
    - E.g. `ip.ecn.changed`
  - States are mutually exclusive within an aspect/subaspect
  - Query language allows wildcarding on the end of a condition name

PTO3 Architecture



PTO3 Key Feature: Provenance

- Provenance must answer two questions:
  - On what raw data does an observation depend?
  - What transformations have been done on that raw data?
- Observations are part of *observation sets*
- Observation sets are created by *analysers*
- Analysers know what other observation sets or raw data files were involved in the creation of a specific observation set
- Analysers store their own commit reference

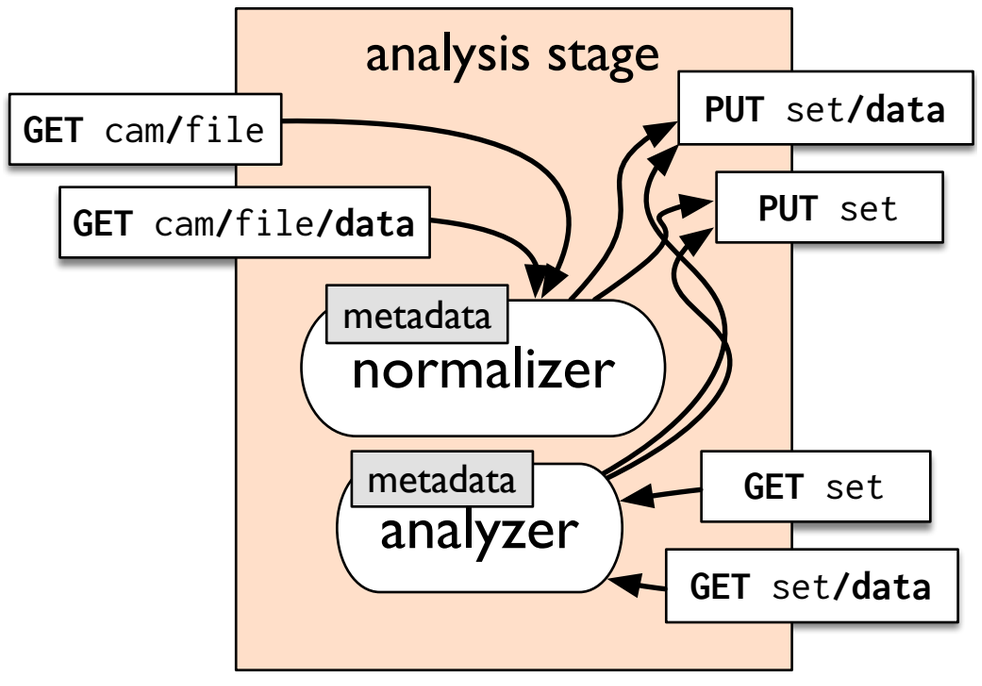


PTO3: Tracebox Data Set

- Send TTL-limited TCP probes, analyse ICMP errors
- Several different Tracebox campaigns run in June 2016
- Result: about 1.5 TiB of JSON data
- Spoiler: even on ridiculously fast hardware, going through 1.5 TiB of data takes some time!
- Solution:
  - normalization is embarrassingly parallel
  - use memory-mapped I/O with aggressive prefetching
  - (do *not* unmarshal JSON, use string operations, *not* REs)

Normalization and Analysis

- Normalizers and analyzers described by metadata explaining how to run them
  - Currently stored in a source repository, not in the PTO itself
- Normalizer/analyzer interface allows any process that can run on UNIX to process data for the PTO → no platform lock-in
  - Normalizer: data on stdin, metadata on fd 3, observations+metadata on stdout
  - Analyzer: observations+metadata on stdin, observations+metadata on stdout
- Not a lot of frameworkiness: tracebox normaliser has just 284 lines of code in Go



PTO3: Tracebox Data Set

Occurrences	Name or Option
9171447313	IP::Checksum
9171446541	IP::TTL
1279130958	IP::DiffServicesCP
326560370	TCP::O::MSS
260492548	TCP::Checksum
172780786	TCP::SeqNumber
21264366	TCP::O::SACKPermitted
5460040	IP::Length
1960762	TCP::Offset
489556	IP::ID
75071	TCP::Window
68811	TCP::O::WSOPT-WindowScale
16568	TCP::O::TSOPT-TimeStampOption
14606	TCP::Flags
13120	IP::ECN
9644	TCP::SPort
8313	IP::Flags
5797	TCP::AckNumber
4646	TCP::UrgentPtr
4143	TCP::Reserved
3403	TCP::O::TCPAuthenticationOption
3172	TCP::O::Echo
3138	TCP::O::CC
2465	TCP::O::CC.ECHO
1335	TCP::O::MD5SignatureOption
1230	TCP::O::CC.NEW
1088	TCP::O::Quick-StartResponse
1055	TCP::O::EchoReply
1037	TCP::O::PartialOrderConnectionPermitted
1028	TCP::O::TCPAlternateChecksumRequest
940	TCP::O::SACK
903	TCP::O::SNAP
864	TCP::O::(null)
828	TCP::O::UserTimeoutOption
682	TCP::O::TrailerChecksumOption
677	TCP::O::SCPSCapabilities
660	TCP::O::TCPAlternateChecksumData
647	TCP::O::PartialOrderServiceProfile
587	TCP::O::SelectiveNegativeAck
526	TCP::O::RecordBoundaries
525	TCP::O::MultipathTCP
458	TCP::O::CorruptionExperienced

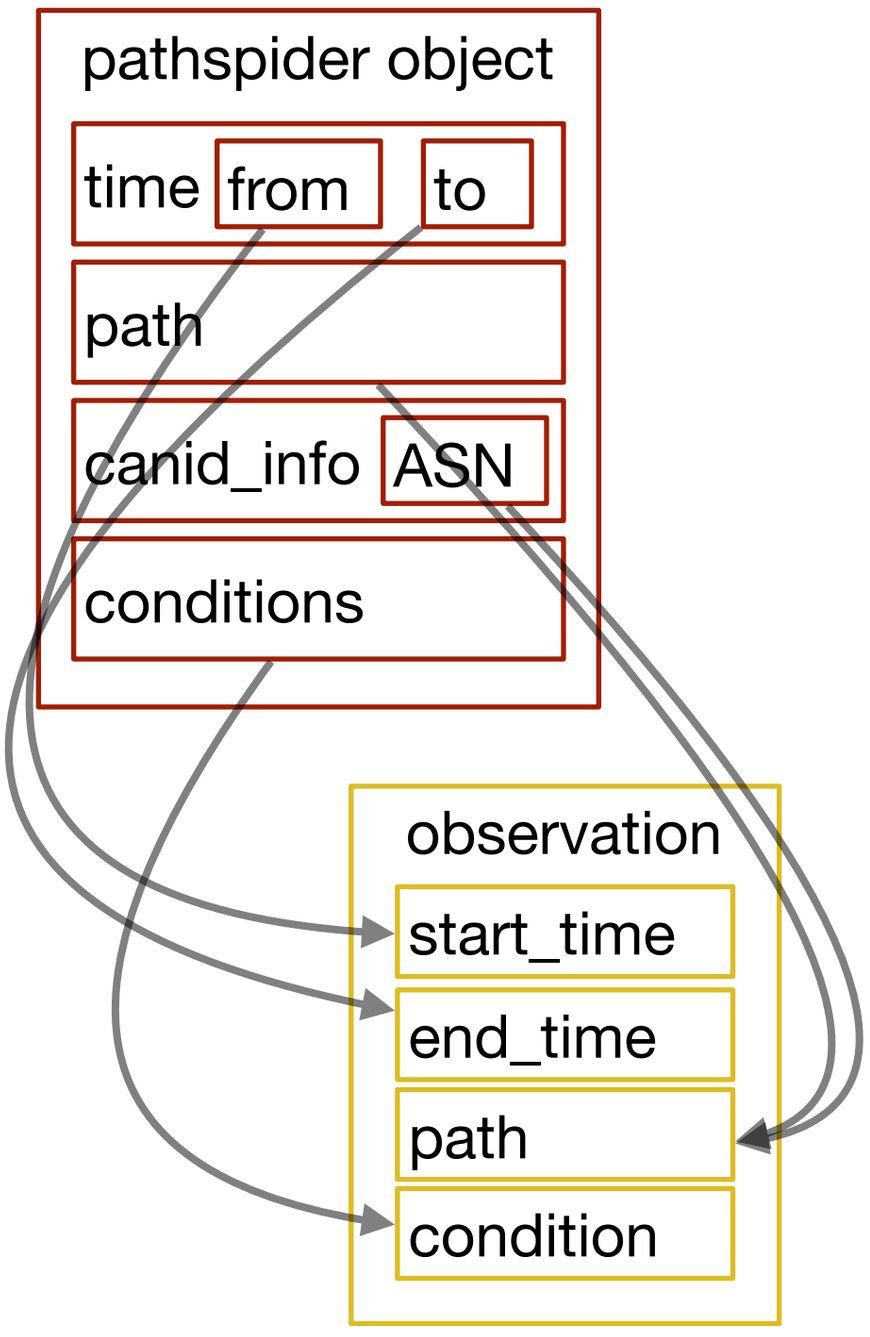
- 9.2 Billion hops
- First time I've had to use a 64-bit counter in anger, yay!
- (This table took 35 min to compute, giving 650 MiB/s throughput)
- We see some very weird middlebox interference indeed!
- Many of these are not suitable as PTO conditions (checksum, TTL)

PTO3: Tracebox Paths and Conditions

PTO3 Condition Name
tcp.option.mss.changed
tcp.option.sackok.changed
tcp.length.changed
tcp.offset.changed
ip4.id.changed
tcp.window.changed
tcp.option.ws.changed
tcp.option.ts.changed
tcp.flags.changed
ecn.ip.changed
tcp.sport.changed
ip.flags.changed
tcp.ack.changed
tcp.urg.changed
tcp.reserved.changed
tcp.option.ao.changed
tcp.option.rfc1072.echo.changed
tcp.option.rfc1644.cc.changed
tcp.option.rfc1644.echo.changed
tcp.option.md5.changed
tcp.option.rfc1644.new.changed
tcp.option.rfc4782.changed
tcp.option.rfc1072.reply.changed
tcp.option.rfc1693.permitted.changed
tcp.option.rfc1146.request.changed
tcp.option.sack.changed
tcp.option.snap.changed
tcp.option.user-timeout.changed
tcp.option.trailer-checksum.changed
tcp.option.scps-capabilities.changed
tcp.option.rfc1146.data.changed
tcp.option.rfc1693.profile.changed
tcp.option.selective-nack.changed
tcp.option.record-boundaries.changed
tcp.option.mptcp.changed
tcp.option.corruption-experienced.changed

- All options are `<feature>.<aspect>`
- E.g. `ip.flags.changed` means that the IP flags have changed.
- Sometimes there are several reasonable alternatives
  - `ecn.ip.changed`
  - `ip.ecn.changed`
- Could have either, chose `ecn.ip.changed` b/c PathSpider compatibility
- If you don’t like a condition name, write an analyser and create your own

PTO3 PathSpider normalization



- PATHspider normalizers even easier to write:
  - Condition comes directly from PS output, some cleanup for v1 conditions
  - Path derived from sip/dip for v1, from path element and canid information in v2
  - written for ECN (but works for all conditions): 338 lines of Go, ~100 of which can be factored out.
- Current work (post summerschool)
  - fuse (fast) Tracebox and (slower) ECN normalizer
  - pull common patterns into Golang support/utility code for writing normalizers/analyzers
  - finish generalizing ECN normalizer for PATHspider.

PTO3 ECN Spider + QoF normalization

- For 2014-2016 datasets, we only have QoF IPFIX output
  - observations derived solely from passive measurement
  - ecn_qof_normalizer extracts conditions from this IPFIX data
- This is a bit more complicated:
  - ecn_on and ecn_off flows have to be matched normally
  - no guarantees about timing, no information about which flows were generated and which captured by chance

PTO3 derived ECN analysis

- 2016/2017 measurements did three simultaneous runs per VP to reduce the noise floor → 2017 ANRW paper
  - ecn_stabilizer combines these into ecn.stable.* conditions that reject noisy samples, grouped by vantage point (taken from vantage metadata key)
- Path dependency can be determined by comparing conditions from multiple vantage points toward the same target
  - ecn_pathdep combines these into ecn.multipoint.* conditions that include path dependency for negotiation and connectivity.

Open issue: performance

- Moved PTO3 to muninn (40-core, 256G RAM, 32T spindle)
- We are primarily I/O bound.
  - Query performance (group/select w/o obset scope) scales ~linearly with number of rows in the database.
  - 500M rows → 12 minute queries.
- Database is unoptimized: we can denormalize and index to go faster, but most interesting queries have to seqscan anyway.

Open issue: Paths

- Issue: two element-for-element identical paths *could* mean different things; practically speaking:
  - "pair paths" have the form [source * target]
  - "hop paths" have the form [source * prehop posthop * target]
  - "trace paths" have the form [source hop hop hop hop target]
- Three ways to reconcile; do path semantics:
  - depend on associated condition?
  - depend on analyzer/normalizer defintion (via metadata)?
  - remain undefined s.t. analyzers/queriers must guess?

Open issue: frontend

- Old web frontend not yet ported to new API
  - ETH has a student working on this
- Python client developed for summer school
  - Dumps data into Pandas dataframes for further analysis
  - We used this client for our IMC submission

Next step: D1.2

- Deliverable due **end June 2018**
- Content to come mainly from IMC submission, PTO/ PATHspider documentation (esp. conditions)

- Brian editing, responsible for PTO sections
- Iain responsible for PATHspider condition sections?