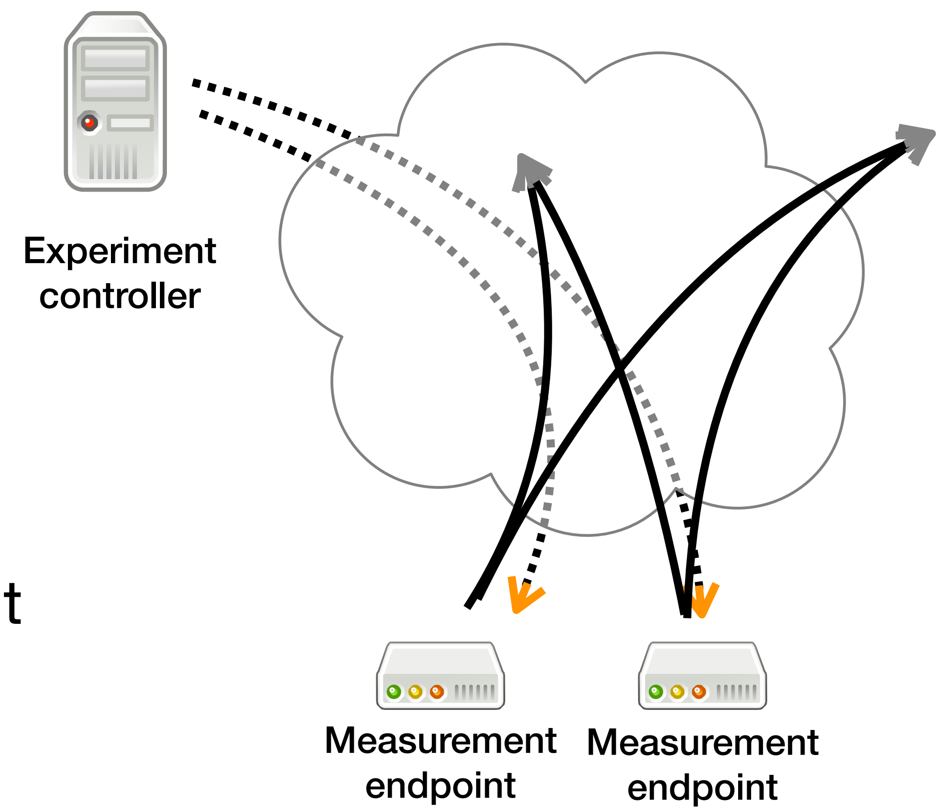
PacketLab:
A Universal Measurement Endpoint Interface

Kirill Levchenko *with*
Amogh Dhamdhere, Bradley Huffaker, kc claffy,
Mark Allman, Vern Paxson

Edge Measurement

- Active measurement from end hosts where vantage point is an experimental factor
  - Censorship and traffic tampering
  - Consumer bandwidth/latency
  - Network topology
- Requires access to measurement endpoints at edge
  - Costly to deploy and maintain



Measurement Platforms

- Dedicated server
  - CAIDA Archipelago (Ark), PlanetLab
- Hardware agent
  - BISmark, SamKnows, RIPE Atlas
- Software agent
  - OONI Probe, ICSI Netalyzr

Measurement Platforms

Measurement Platforms

- Dedicated server
  - CAIDA Archipelago (Ark), PlanetLab
- Hardware agent
  - BISmark, SamKnows, RIPE Atlas
- Software agent
  - OONI Probe, ICSI Netalyzr

Obstacles to Sharing

- **Compatibility**
  Each platform has its own usage model and API, experimenter must port experiment to each one
- **Incentives**
  Operator bears some of the costs of outside experiment
- **Trust**
  Operator must trust experimenter or verify each experiment

Obstacles to Sharing

- **Compatibility**
  Each platform has its own usage model and API, experimenter must port experiment to each one
- **Incentives**
  Operator bears some of the costs of outside experiment
- **Trust**
  Operator must trust experimenter or verify each experiment

How do we lower barriers to sharing?

PacketLab Overview

- Light-weight universal endpoint interface
  - Write experiment once, run anywhere
  - Easy to port to new platforms
- Remove platform operator from experiments
  - Shifts cost of experiment to experimenters
- Give platform operators fine-grained control over allowed outside experiment behavior
  - Reduces burden of trust between operators and experimenters

Disclaimer

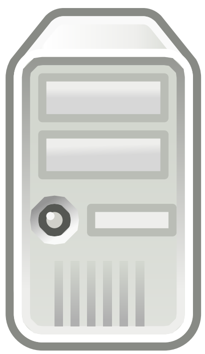
- Not a new measurement platform
- Complements (does not replace) existing interfaces
- Single point in large design space
  - Want to get critical feedback and stimulate discussion
- Preliminary design, not a finished product
  - Alpha-quality proof of concept prototypes

Key Technical Ideas

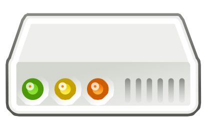
- Move experiment logic from network endpoint
- Use certificates for access control
- Endpoint-experimenter rendezvous
- Monitor programs define allowed experiment behaviors

Traditional Endpoint Model

Experiment Controller



Control logic



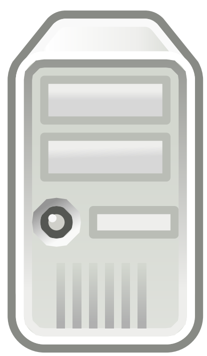
Endpoint

Experiment logic

Network interface

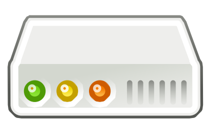
PacketLab Endpoint Model

Experiment Controller



Control logic

Experiment logic

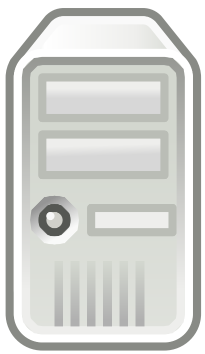


Endpoint

Network interface

PacketLab Endpoint Model

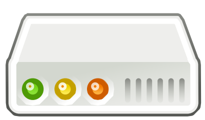
Experiment Controller



Control logic

Experiment logic

PacketLab Interface



Endpoint

Network interface

PacketLab Endpoint

- PacketLab endpoint ==
  VPN endpoint with measurement knobs and dials
- TCP/UDP sockets and raw IP I/O (where available)
- Compatible with multiple deployment regimes
  - Software agent, hardware agent, dedicated server
- Minimal assumptions about underlying hardware
  - Easy to support PacketLab interface on endpoints

Endpoint API

- Resembles Berkley sockets
- Controller schedules packet to be sent immediately or at future time (`at_time`)
- Controller polls for received packets (`npoll`)
  - Packets not forwarded to controller immediately
  - Allows controller to manage access link load

```
nopen(sktid, proto)
nopen(sktid, proto, locport,
      remaddr, remport)

nclose(sktid)

nsend(sktid, tidx, at_time, data)

npoll(sktid, until_time)

ncap(sktid, filt, until_time)
```

Endpoint Information API

- Need to provide some endpoint information to controller
  - Endpoint IP address, current time (endpoint clock), etc.
- Exported via *endpoint memory space*
  - Analogous to hardware device registers
- Accessed via endpoint API
  - **`mread(addr, bytecnt)`** *and* **`mwrite(addr, data)`**
- Structure of memory space and addresses of values defined by PacketLab API

Experiment Controller

- Tells endpoints exactly …
  - What packets to send and when
  - Which packets to capture
- Run by experimenter, *not* endpoint operator
  - Shifts cost from operator to experimenter
- Ephemeral: exists for duration of experiment only
- Needs to implement all protocols used in experiment

Rendezvous

- Experiments distribution on *pull* model:
  Endpoints contact experiment controllers for experiments
  - Endpoints need a way to find experiment controllers
- **Rendezvous server:** Directory of active experiments
- Experimenters *publish* experiments to rendezvous server
- Endpoints *subscribe* to (i.e. poll for) experiments
- Need a handful of community-operated servers
  - Like NTP, DNS, or PGP servers

Access Control

- Operators give experimenters *digitally signed certificates* granting access to their platform (endpoints)
  - Out of band, based on operator's specific policy
- Each endpoint has a root of trust (set of public keys)
  - Only agrees to do experiment signed by a trusted key
  - Operators install their key when they deploy endpoint
- Experiment controller provides certificate to each endpoint to prove it is allowed to do experiment
  - Certificates can be chained for delegation
  - *No direct communication between operator and endpoint*

Control of Experiments

- Operator will want to restrict the kinds of experiments and experimenter can run on endpoints
  - Today this is based on trust relationships
- Operator specifies *experiment monitor program* that defines what packets experimenter can send during experiment
  - Interpreted program encoding fine-grained access control policy
  - Similar to BPF, but need slightly richer mechanism
- Monitor program attached to experiment certificates
  - Presented to endpoint with certificate
  - Part of signed certificate (verified to be from operator)

Monitor Program

- Executes in a restricted VM (like BPF)
- VM memory space = endpoint memory space
  - Accessible using **`mread`** and **`mwrite`**
- Written in a C-like language, compiled to bytecode
- Certificates contain compiled bytecode of monitor

Monitor Program

```
in_addr_t ping_dst = 0; // destination of traceroute

uint32_t send(const union packet * pkt, uint32_t len) {
    if (pkt->ip.ver == 4 && pkt->ip.ihl == 5 &&
        pkt->ip.proto == IPPROTO_ICMP &&
        pkt->ip.src == info->addr.ip &&
        pkt->ip.icmp.type == ICMP_ECHO_REQUEST)
    {
        return len; // allow
        ping_dst = pkt->ip.dst;
    } else
    return 0;   // deny
}
```

Monitor Program

```
in_addr_t ping_dst = 0; // destination of traceroute

uint32_t send(const union packet * pkt, uint32_t len) {
    if (pkt->ip.ver == 4 && pkt->ip.ihl == 5 &&
        pkt->ip.proto == IPPROTO_ICMP &&
        pkt->ip.src == info->addr.ip &&
        pkt->ip.icmp.type == ICMP_ECHO_REQUEST)
    {
        return len; // allow
        ping_dst = pkt->ip.dst;
    } else
    return 0;   // deny
}
```

Structure in endpoint memory space, accessed in monitor program as struct

Monitor Program

View of IP packet as a struct/union

```
in_addr_t ping_dst = 0; // destination of traceroute

uint32_t send(const union packet * pkt, uint32_t len) {
    if (pkt->ip.ver == 4 && pkt->ip.ihl == 5 &&
        pkt->ip.proto == IPPROTO_ICMP &&
        pkt->ip.src == info->addr.ip &&
        pkt->ip.icmp.type == ICMP_ECHO_REQUEST)
    {
        return len; // allow
        ping_dst = pkt->ip.dst;
    } else
    return 0;   // deny
}
```

Structure in endpoint memory space, accessed in monitor program as struct

Monitor Program

```
uint32_t recv(const union packet * pkt, uint32_t len) {
    if (pkt->ip.ver == 4 && pkt->ip.ihl == 5 &&
        pkt->ip.proto == IPPROTO_ICMP && (
            (pkt->ip.icmp.type == ICMP_ECHO_REPLY &&
             pkt->ip.src == ping_dst) ||
            (pkt->ip.icmp.type == ICMP_TIME_EXCEEDED &&
             pkt->ip.icmp.orig.ip.src == info->addr.ip &&
             pkt->ip.icmp.orig.ip.dst == ping_dst)))
        return len; // allow
    else
        return 0;   // deny
}
```

Monitor Design Options

- C-like custom language
  - Familiar to programmers
  - Can restrict language features to match model
- P4 dataplane programming language
  - Existing toolchain support
  - Parse arbitrary protocols
- Same bytecode representation

Encouraging Sharing

- PacketLab defines mechanism, *not* policy
- Super-secret subversive goal:
  - Make PacketLab attractive even if you don't want to share …
  - … so you have no excuse *not* to share later
- PacketLab *project* may try to encourage sharing
- PacketLab *protocol* is the mechanism for doing so

Where We Are Today

- IMC 2017 short paper
- Interest from experimenters
- Interest from platform operators
- Working on reference implementation
  - For Unix-like operating systems

PKTLAB

Conclusion

- **PacketLab:** an universal interface to network measurement platforms (endpoints)
- Value proposition for **experimenters:** a single interface to multiple measurement platforms
  - Write experiment once, run anywhere
- Value proposition for **platforms operators:** gives experimenters *controlled* access to your platform