Active measurements needs

During the week of May 1-5, CAIDA hosted an in-person workshop at SDSC, where we discussed active measurement needs of the community as we planned for the future of Ark and measurement infrastructure in general. The feedback that we received was wide and varied; we discussed raw measurement primitives and capabilities desired by the community, and technical solutions that we could pursue that would support the community.

We asked participants to describe the raw measurement primitives that we should strive to support from active VPs. These include:

(Figure 1: Candidate list of raw measurement primitives)

1. HTTP
2. DNS
3. custom TCP packets
4. sourcing of packets with spoofed addresses
5. BGP announcements
6. topology measurements (ping, traceroute, alias resolution)
7. TLS handshakes and STARTTLS
8. packet capture of unsolicited traffic received by the VP
9. bandwidth measurements
10. zgrab-style banner grabs
11. arbitrary forwarding of packets delivered to the VP by a service

These raw measurement primitives provide researchers the building blocks to build more complex measurements, such as DNS queries that obtain sets of authoritative nameservers in the resolution path of a domain name that could then be followed with ping and traceroute measurements to identify how close those nameservers are to the resolver, and the path that the resolver takes to the nameserver. These raw measurements overlap; a user could construct their own TCP-based traceroute with custom TCP packets, for example. However, our job is to support the types of measurements that researchers typically use, and provide a degree of flexibility in how the researcher obtains active measurements that answer their questions.

Current capabilities

Currently, Ark regularly conducts ping, traceroute, and alias resolution measurements, and has supported occasional research by the community that required HTTP, DNS, custom TCP packets, or bandwidth measurement, as well as a prototype service (PacketLab) that supports arbitrary forwarding of packets delivered to the VP by an external measurement service.

Currently, Atlas supports regular ping, traceroute, DNS queries, NTP, and geolocation through ping measurements. Periscope supports ping and traceroute measurements from (a dwindling list of) public looking glasses. EdgeNet supports arbitrary measurements within containers provided by researchers, and conducts their own ongoing Internet-scale topology measurements. Finally, M-Lab supports bandwidth measurements (NDT), topology measurements towards systems that contact M-Lab servers, and supports community projects (reverse traceroute, and WeHe) by providing server infrastructure for those projects.

Participants expressed concerns about possibilities that some of the new list of primitives could be misused in a way that causes harm to VP hosting providers or measurement targets.

For example, a host in a country that censors HTTP, DNS, or TLS could be harmed by measurement traffic that contains keywords that trigger the censor. Similarly, spoofed packets can be used both to test source address validation (SAV) deployment, and for denial of service attacks. To support the candidate list of raw primitives above, we need to revisit the Ark memorandum of cooperation with site hosts, and implement mechanisms to prevent or limit measurements that could be problematic for the site host, and allow site hosts to opt-out of measurements that they do not want to support.

Spectrum of Solutions

In terms of technical solutions that we could pursue, participants recognized that the approaches we could consider exist on a continuum (from least to most restrictive):

- Full shell access on the probe
- run measurement code in a container on the probe
- Domain specific language (DSL) to run tests, send packets, do logic

- VPN access to send packets from probe, do logic elsewhere / no logic
- API to run tests, send packets, do logic elsewhere / no logic
- No access, just use provided data

Current data access capabilities

Currently, Ark provides full shell access on the probe to vetted researchers, and an API (Vela) to vetted researchers to run selected measurements. We also provide researchers restricted access to data resulting from CAIDA's own measurement campaigns that use Ark, under acceptable use policies (AUPs) to protect and sustain the infrastructure. Data older than one year is available to the public.

Atlas provides an API to run tests, and has a minamalist AUP that requires a user to have Atlas credits to schedule a measurement. This minimalist AUP is is appropriate because the types of measurements that an Atlas VP can conduct has been limited by RIPE NCC to traceroute, ping, DNS, and NTP. Atlas provides public access to the data that Atlas VPs have collected.

Finally, EdgeNet provides the ability for vetted researchers to upload containers that can contain measurement experiments, as well as access to data that EdgeNet experimenters make available. EdgeNet's AUP requires end-users to consider if a network administrator at the end-user's institution would allow the experiment if it occurred at their institution. Beyond this, EdgeNet operates using a high-trust model; they vet the academic supervising an end-user, and rely on the academic to ensure the supervised end-user behaves reasonably.

Participants were particularly excited in further development of two of these aspects: "run measurement code in a container on the probe" and "Domain specific language (DSL) to run tests, send packets, do logic".

Design and prototype plans for container approach

For the container approach, we plan to prototype an approach that uses containers and key features provided by Linux: IP tables, network namespaces, and control groups (cgroups). We hypothesize that these features will allow us to restrict measurement code in a

container from exhausting available resources on the node, and enforce simple requirements that a site host prescribes, such as which protocols they allow and how much bandwidth they are prepared for the measurements to consume. This approach fits the desires of many researchers: the ability to develop and test their measurement code on their own systems, and then run that code on third-party platforms once they believe the code is ready. The challenge remains that a container approach does not necessarily prevent measurement activities that the host might not support, such as DNS or HTTP measurements for websites that might be censored by their country.

On the positive side, a container appraoch would allow any programming language installed on the measurement platform to be used to perform measurements, allowing researchers to use the language they are most familiar/comfortable with. The language can be used to its fullest extent, e.g. libraries, flow control structures, and provides researchers the ability to use existing protocols in any way, or add new protocols as desired (limited by site restrictions).

However, a container approach would require more programming, networking, and operating system knowledge from researchers in order to perform measurements, as lower level abstractions can make programming more complicated. As platform operators, we would become heavily reliant on the container environment and the operating system providing effective ways to restrict behaviour, e.g. sending certain types of packets, packet rates, reading/writing files on local disk, or exhausting CPU time. As a platform operator, we would also need to understand and cover everything that might be a security issue, as the containers could do anything allowed by the environment that we configure. Further, operating system updates may change or break the mechanisms we use to restrict measurements.

Design and prototype plans for domain-specific approach

We discussed the domain-specific language approach as well; game engines often provide a Lua-based interface that allows players to create custom extensions that use existing functionality in the game engine to enhance the game player's experience. For Internet measurement, one possible design would provide an interface that allows researchers to place logic on the VPs to extend a deployed measurement engine to allow for measurement logic to react to measurement results as the node collects them. Participants described two motivating examples from the DNS space: monitoring when DNS

operators deploy a new zone file via fine-grained monitoring of SOA records contained in the zone file, and monitoring of authoritative resolvers for domains under attack. The APIs exposed by Atlas hamper these measurements, as the API does not allow measurements to quickly react to results as VPs collect data. We plan to prototype a python-based DSL that provides a convenient interface to scamper (Ark's measurement engine) processes on Ark nodes.

Our immediate goal with a DSL is to provide a convenient interface for DNS and ping measurements; participants at the workshop described their challenge in quickly obtaining and measuring the resolution path for a given domain name. We plan to add DSL support for other measurement primatives once we have prototyped the infrastructure to support this initial use case.

On the positive side, a DSL can provide high level abstractions for common networking tasks that could otherwise consist of multiple complicated steps, e.g. traceroute. A DSL is easier for non-programmers to write useful measurement programs, with less knowledge of things like socket APIs. Further, the implementation of measurement primitives is likely better than the average roll-your-own version, and it is harder to do "bad" things if the language doesn't provide a way to do them, e.g. send certain types of packets, too many packets, read or write to files on the local disk, exhaust CPU time.

However, a DSL must be still be learned by a user before they can effectively use it. Even if it is based on top of an existing language (e.g., Lua, Python, Perl) there will be new domain specific parts. The limited scope or cut down nature of the DSL might make it difficult to perform some tasks that could be done easily in a general purpose language, e.g. if the DSL restricts looping constructs. Researchers are reliant on DSL maintainers exposing useful measurement primitives, and the ways the DSL allows them to be used/configured, e.g. setting fields in headers, sending certain types of packets. Researchers are reliant on the maintainers of the DSL keeping it up to date, e.g. to work on modern systems, add new protocols.

Monitor specification report.

Our current Ark infrastructure has taught us valuable lessons about how we should provision hardware and software. Key lessons include:

1) select a single operating system that allows for scalable
system administration.
2) use the capabilities of the node to prevent premature wear.

For the current Ark system, now over 10 years old, we manage a collection of Raspberry Pi devices running Raspbian (primarily versions 10 and 11 on Raspberry Pi 2 and 3 devices) and 1U rackmount devices running FreeBSD 10.3 for the i386 architecture. Software installation and maintenance is currently primarily a manual job: if we wish to deploy or update software, we have to push the software out to the VP. It is extremely challenging to maintain this platform because we have to revisit which VPs have what software, as we can only push software to VPs that are up and reachable; not all VPs are up and reachable at any one time.

Further, the FreeBSD 10.3 nodes are nearly end-of-life: i386 has become a tier-2 platform since FreeBSD 13. We are therefore planning to settle on debian-based (i.e., debian, raspbian, and ubuntu) OSes going forward -- both with physical node deployments and future containerized deployments that we have planned.

Key towards future scalable maintenance is to work toward a pull-based model, where we publish packages that contain measurement software, and the vantage points self-update their own packages when they are operational without system administrator intervention. We have settled on debian-based systems because we can publish debian packages for them. Further, these packages can serve as input for future docker containers, allowing volunteers to provision Ark nodes in interesting networks, including U.S. R&E networks.

Finally, we have learned that we need to minimize write activity to SD cards, which can fail because each write incurs wear on the card, significantly shortening the lifetime of otherwise usable vantage point hardware. To minimize SD card wear, we have provisioned and deployed test VPs with (128MB) /ramdisk partition to store in-progress team-probing traceroute data. This ramdisk size is appropriate for the Raspberry Pi VPs that have 512MB of RAM, as it still allows 384MB of RAM to run other software on the VPs, but is a constraint for activities other than team probing. For example, our current IPv4 prefix-probing measurements require up to 200MB of storage (more than the available space on the ramdisk), and our team-probing measurements require up to 105MB of storage (nearly all of the space on the ramdisk), requiring us to continue to use SD card storage for these activities. Further, we wish to be able to support the on-VP storage needs of third-party researchers without wearing out deployed SD

cards. Therefore, we plan to deploy nodes with a minimum of 4GB of RAM in the future. Doing so will allow us to provision significantly larger ramdisks (approx 2GB).