Instrumentation Thoughts

GENI Measurement Workshop

Neil Spring

University of Maryland

Ramblings of a GENI newbie

The Outline:

- My context
  - It never occurs to me to assume control over the entire network.
  - But I feel entitled to instrument my own traffic.
- Three key problems to address in supporting measurement*
- Uncooked idea for feedback

*Apologies to those way ahead of me.

My Context (measurement projects)

- **Scriptroute**
  - Script the logic (smarts) of active measurement, hide the systems stuff
  - Permit remote execution on PL since 2002
- Reverse engineering
  - **Rocketfuel**, **Discarte** synthesize network path information to get a more complete or more accurate picture for simulation
- Diagnosis
  - **Tulip** - find the links to blame
  - **Serenity** - try to find the wireless errors, instead find the wireless tracing errors.

Three Key Requirements

1. Support for dynamic summarization and filtering
2. Precise and prompt, standardized timestamping
3. Ability to compensate for virtualization

Summarization / Filtering

- Imagine assembling “complete” information:
  - Timestamps for every packet at every step.
  - Intractable to collect without reducing data.
- Difficult to predict what’s needed (to construct generic measurement).
- *That means distributing code to summarize and filter*
- (bpf is a good example; CoMo; Skitter’s compact trace representation)

Precise and Prompt, Standardized Timestamps

- Currently quite difficult to explain where timestamps come from, and they’re not very good.
  - Often measure non-network activities
- *Design timestamps on the forwarding path*
  - Use hardware, note the source!
  - Don’t need synchronization (that’s software).
    - NTP adjustments troublesome (see Darryl Veitch’s talk... awesome.)
  - Define time to be public (even from NTP clients).

Compensates for virtualization

- Don’t need to see traffic that belongs to others, but...
- Should be able to tell that other activities are interfering, in the absence of complete isolation.
- *Include opaque “other traffic” in traces and counters*.

*Inspired by the “top” argument on PlanetLab before good memory resource control.

Idea in development

- $>$80% of the time when my code breaks, I look at a stack trace.
  - unhandled ruby exceptions
  - gdb after abort()/assert()/(*NULL)
- What if network error reports were so descriptive and uniform?
  - could applications (windows, firefox) quickly help users repair?
  - could applications repair more errors on their own?

Network Stack Trace

- Easy example:
  - arp timed out from router 128.8.126.1
  - ip host unreachable from 128.8.126.1
  - tcp connect failed
- More stuff to include:
  - include unplugged cable? power off?
  - unresponsive but (802.11) associated?
  - queue too long? effects on groups of packets?
- Can we expose errors at each layer to be propagated and handled?