

Experiences With NIMI

Andrew K. Adams and Matt Mathis

National Laboratory of Applied Network Research (NLANR)
Pittsburgh Supercomputing Center
{akadams,mathis}@psc.edu

Vern Paxson

ICSI Center for Internet Research (ICIR)
and
Lawrence Berkeley National Laboratory
vern@icir.org, vern@ee.lbl.gov

SAINT 2002
February 1, 2002

Key facets of NIMI measurement infrastructure:

- Designed to work across domains, by emphasizing full control of platforms owned by a domain ...
- ... but also ease of delegating some measurement functionality, in a secure fashion.
- Not about a particular type of measurement, but rather facilitating a potentially large class of measurements

Performing measurement:

An infrastructure is built by installing NIMI probes (nimid) throughout the network. All NIMI's within the same administrative domain are configured by the domain's Configuration Point of Contact (CPOC).

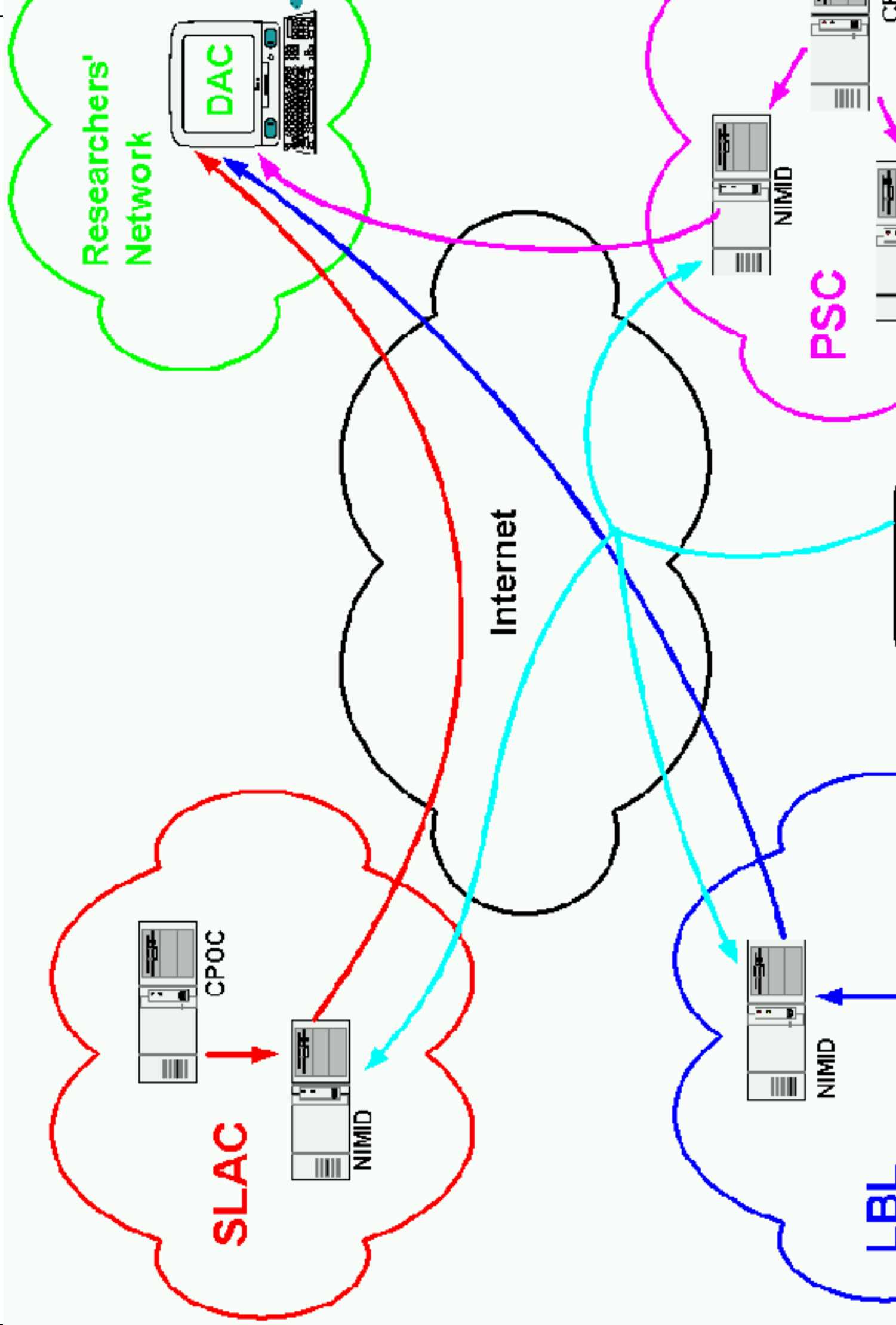
User invokes the Measurement Client (MC) to schedule a set of measurements at some point in the future (including "now").

Performing measurement, con't:

MC contacts different NIMI's to scheduled measurement.
Tells platforms where to send the results (DAC).

Data Analysis Client (DAC) eventually receives results
(possibly on another host).

User at some point uses MC to delete results from NIMI
platforms.



NIMI current status:

14 measurement tools supported --

- abcd - CDN benchmark
- cap/capd - measures bulk transfer capacity
- dig - dns resolver
- discardd/traffic - generic traffic sender/discarder
- fsd - measures ICMP generation time
- ftp - file transferrer
- httpperf - http transferrer
- imapper - topology mapping
- mflect - multicast reflector
- mtrace - multicast route viewer
- traceroute - ip route viewer
- treno - measures bulk transfer capacity
- wget - http transferrer
- zing - measures tcp loss & delay variation

NIMI current status, con't:

Code base:

- 65K lines of C
- runs under FreeBSD, NetBSD, Linux, Solaris

Operational at 50+ sites --

- 38 North America, 9 Europe, 3 Asia, 2 South America
- diversely administered
- chronic outages due to size + sundry failures

Current architecture at its administrative scaling limit ...

Lessons learned:

During the last two years, we have discovered that:

- NIMI is a great resource ...
- ... but extremely hard to implement!

Our ubiquitous measurement infrastructure was impeded by 3 classes of problems ...

Lessons learned, con't:

Design shortfalls:

- incorrect, or omissions in functionality
 - remote error handling
 - tracking measurements
 - data integrity
 - URLs
 - non-blocking I/O (backlog, SO_ERROR)
 - multiple TCP connections
 - retry congestion
 - ACL granularity
 - key distribution (authentication)

Implemented in software, so correctable.

Lessons learned, con't:

Heterogeneity issues:

- remote administration headaches
 - initial ssh access
 - tools requiring root privileges
 - modification to system configuration (RFC 1323)
 - crypto libraries (RSA/RSAEuro incompatibilities)
- platform & administrative inconsistencies
 - kernel robustness (mbufs & misbehaving tools)
 - inconsistent "standard" measurement tools
 - missing system tools (Perl)

Attempt to encapsulate areas of differences (though an uphill battle.)

Lessons learned, con't:

Large systems complications:

- incongruence within large distributed systems
 - clocks
 - DNS flakiness
 - subtle APIs
 - exhausting resources
 - firewalls/filtering
 - multicast
- routine maintenance headaches
 - SSH & NTP security patches
- managing large-scale studies
 - sheer volume of data

Leads to incremental design refinements.

A great resource:

Good news, we got the majority of the architecture right!

Used for several large-scale measurement studies:

- "constancy" study captured 273M pkts between 914 hosts pairs, plus 48,600 bulk transfers (IMW 2001);
- "striped unicast" study analyzed MINC-like unicast traffic between 35 sites (in submission to TON);
- "bulk transfer capacity" tests calibrated different tools for measuring TCP throughput (IMW 2001);
- "content distribution network" study measured 300,000 Web transfers (IMW 2001);
- "ICMP generation times", study measured ICMP generation times from routers (PAM 2002);

NIMI is a great resource, con't:

- "iMapper", follow on to Mercator;
- "Isobar", study abstracting network into "equivalent performance" regions (likely SIGCOMM 2002 submission).

As well as on-going MINC inference, multicast connectivity, FTP, traceroutes, treno measurements.

Future directions (or cracking the administrative barrier) ...

Rethinking NIMI certificates:

Current NIMI certificate software is home-grown:

- based on RSAREf/RSAEuro
- requires custom management (no public key servers!)
- trust that we didn't botch it
- keys map to NIMI's and MC

Instead: use X.509 certificates (over OpenSSL):

- certificates (keys) map to individual people
- long-lived
- existing signing & distribution servers (e.g., Verisign)
- run a Certificate Authority that issues measurement group certificates on reception of a correct X.509 certificate
- shift certificate management over to standard framework familiar to admins

1st hard problem to solve -- secure upload:

Experience shows that large measurement projects frequently need to update measurement tools ("user" code).

Supporting diverse measurements also needs this.

Problem: how can you trust the code?

Threats:

- subverting the platform (use it as stepping stone)
- using NIMI for attacks (e.g., distributed DoS)
- affecting integrity of other measurements

2nd hard problem to solve -- resource management:

Current granularity of control doesn't address concurrent measurements.

Need fine-grained access to resources such as:

- CPU, memory, disk space
- network utilization
- well-known ports
- packet filter
- destination (e.g., porn protection)

Observation: if you can solve this problem, you probably have the right hooks to solve the secure upload problem, too.

Possible solutions:

Need to protect system & network from corrupt tools.

Simple solutions:

- "trust-the-researcher" model doesn't scale
- ask CPOC admins to hand inspect code doesn't scale

Sandboxing:

- enforcement daemon
- network proxy
- "safe" languages
- link against trusted library
- kernel modifications

Possible solutions, con't:

Enforcement daemon (monitoring/policing):

- detect unaccounted system resources
- relatively cheap to implement
- but only after the fact
- not good enough to prevent malice

Network proxy (generic packet filter & raw I/O daemon):

- solves packet filter woes
 - insufficient devices
- sender latency & receiver time-stamp problems
- can be subverted

Reasonable protection for many classes of failures ...

Possible solutions, con't:

Safe languages:

- examples "Java", "Python", Perl taint
- restricted programming semantics
- restricted semantics for packet I/O
- complete control over resources

Trusted library:

- complete sandboxing
- requires replacing libc on all hosts!

Kernel modifications:

- requires kernel patches & builds!

Perhaps: different solutions for different environments?

In Conclusion:

- Translation to C++ to enhance maintainability (near-term).
- Replace current messaging with XML to allow inter-operability (near-term).
- Replace current authentication framework with X.509 over OpenSSL (near-term).
- Tackle secure upload and resource management:
 - packet capture & sending (pcapd)
 - need language for specifying resource access & control policies
- Support for adaptive measurement.
- GIMI (Global ... in collaboration w/ Paul Barford, UWisc.)