The Design and Operation of "mon"

Jim Trocki
Transmeta Corporation
trockij@transmeta.com

http://www.kernel.org/software/mon/

Overview

- **What is ‘mon’?**
- **Server responsibilities**
- **Monitor responsibilites**
- **Alerts**
- **Alert management**

Overview (contd...)

- Clients and their function
- Configuration details and examples
- Example extensions
- Interesting applications
- Experience

What is "mon"?

"mon" is a tool for monitoring the availability of services and applications.

- **Used by NOCs and IT staff for fault detection and alert management. For example:**
  - **Send an alphanumeric page to NOC staff when routing horked**
  - **Submit trouble ticket when an application becomes inoperable**
  - **Record the routing history between HQ and a branch office, send notification when path changes**
- **Written in Perl**
- **Distributed under GNU General Public License v2**

Features of mon

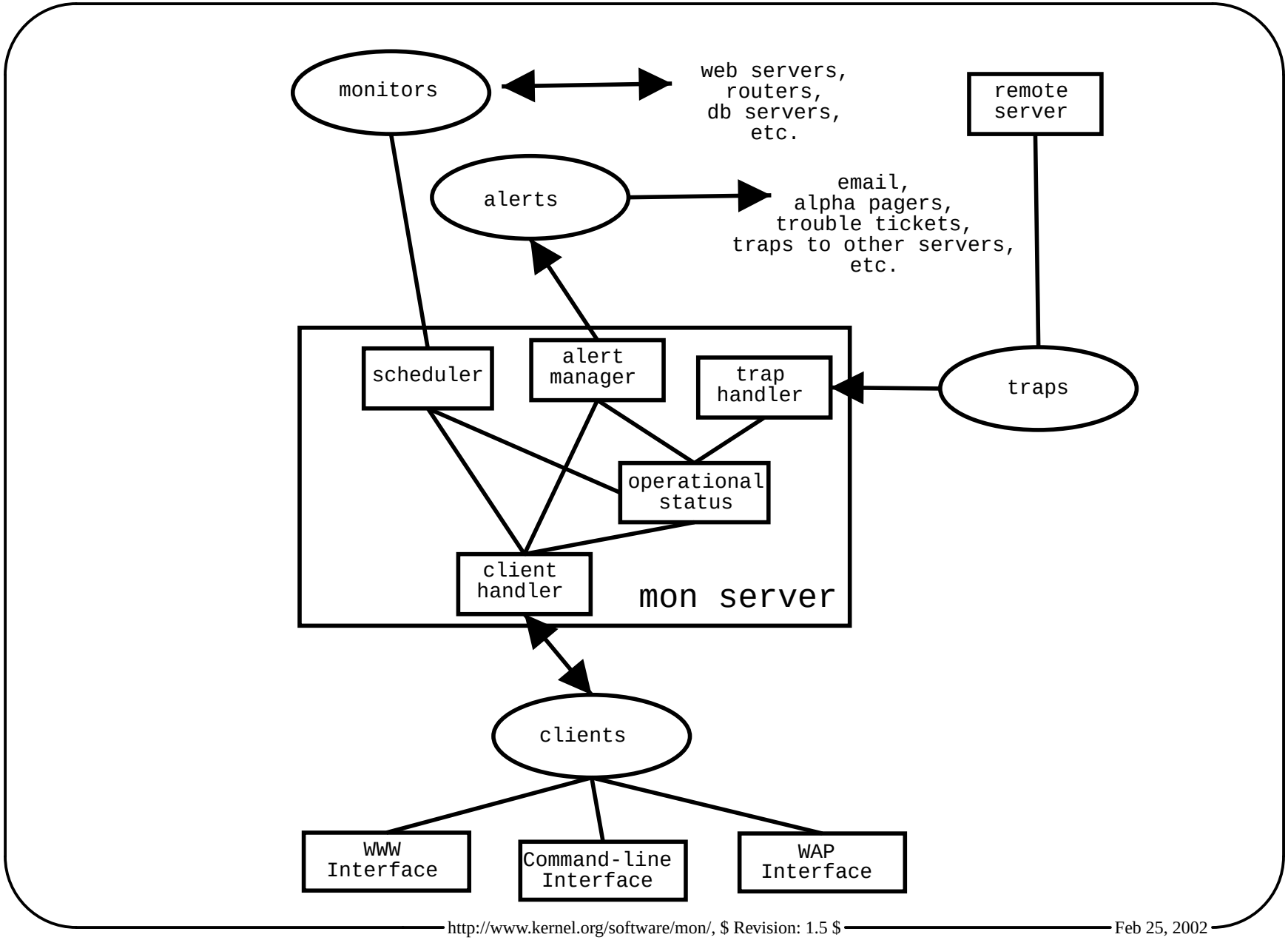
- **Portable (thanks to Perl)**
  - **Linux, Solaris, BSD, Cygwin, ...**
- **Simple yet very adaptable design**
- **Can monitor anything, no clients required**
- **Configurable, extensible**
- **Supportive community, mailing list mon@linux.kernel.org**

Design Goals of "mon"

- **Simple to add alerts and monitors**
- **Simple way of cross-connecting tests and alerts**
- **Simple way of gathering data for report generation**
- **General-purpose, if you can test it with software, you can monitor it**

Components

- **Server**
  - **Schedules monitors, handles traps, alerts, clients, logs.**
- **Clients**
  - **Query and control the server**
- **Monitors**
  - **Communicate with monitored systems via HTTP, SNMP, etc.**
- **Traps**
  - **Send notifications from remote systems**
- **Alerts**
  - **Perform actions on failures, page, email, etc.**


monitors
web servers,
routers,
db servers,
etc.
remote
server
alerts
email,
alpha pagers,
trouble tickets,
traps to other servers,
etc.
scheduler
alert
manager
trap
handler
traps
operational
status
client
handler
mon server
clients
WWW
Interface
Command-line
Interface
WAP
Interface

Server Responsibilities

- **Schedule tests**
  - **Run monitors when necessary**
  - **Gather output and exit status**
- **Accept remote traps**
- **Serve clients**
  - **Deliver operational status**
  - **Accept control commands**
- **Manage Alerts**
  - **Suppress repetitive alerts**
  - **Alert only during specified time periods**
  - **Evaluate dependencies**

Configuration File

- "Cross-connects" monitors to alerts
  - Think of telephone switchboard or patch panel
  - Any monitor can be wired to any alert
- Defines what is to be monitored and how
  - What monitors are to be used
  - What hosts are to be monitored
  - What services are to be monitored
- Defines when alerts happen
  - On which failure
  - On which failure->success transition
  - Frequency
  - Time of day

Monitors

- Test the condition of a service
  - Usually one service test per monitor
  - Tests are user-definable
  - SNMP, HTTP, SMTP, ICMP echo, etc.
  - Application-level tests possible
- Report summary and detailed results
- Exit reporting success/failure

Monitors cont'd

- Written in an arbitrary language
  - Most are in Perl, /bin/sh
  - May call third-party software
  - No binary linkage with mon itself
  - Independent from the mon server
- Invoked as separate processes
  - Many may be run in parallel
  - Hundreds may run in a minute
- Short-lived
  - Start, test, report, exit
  - Helps minimize impact of memory leaks
- Simple to write

Many Available Monitors

Numerous server tests

- **http, lpd, smtp, ldap, imap, pop3, telnet, dns, disk quotas, netware**
- **msql, mysql, oracle, postgres, informix, sybase**
- **reboot, processes, rpc, clock, disk space, RAID**
- **Brocade fcal switches, traceroutes, router interfaces, ipsec tunnels, Foundry router chassis, bgp, RADIUS**
- **Compaq chassis, NT services, samba, printers**

Traps

- Traps are notifications sent to a mon server from an external entity
  - another mon server
  - a stand-alone probe
- Contain the same information as passed by monitor scripts
  - summary
  - detail
  - exit status
- Allows distributed mon agents to send their status to a centralized mon server

Alerts

- **Report the failure status detected by a monitor**
- **Independent from the mon server**
- **Accept input from the mon server**
- **Invoked as separate processes**
- **Written in any language**
- **Simple to write**

Available Alerts

- E-Mail
- SNPP (alphanumeric paging via TCP/IP)
- Qpage (alphanumeric paging via modem and TAP/IXO)
- Trap to other mon server
- AIM
- Bugzilla
- GNATS
- HP Openview
- SMS
- WinPopup
- NetApp snap delete

Alert Management

- **Alert decision logic in the server**
- **Squelch repetitive alerts**
  - **time period**
  - **alertafter num**
  - **alertafter num timeval**
  - **alertafter timeval**
  - **alertevery**
  - **numalerts**
- **Dependencies**
  - **If router is down, don't alert for unreachable things beyond it**
  - **A simple first-pass at root-cause analysis**
  - **Dependencies are Perl expressions**

Time::Period Specifications

Time::Period by Patrick Ryan

- True or false if a time(2) is within a specific period
- scale {range [range ...]}
  - scales: yr, mo, wk, yd, md, wd, hr, min, sec
  - ranges: Mon-Fri, 1-365, 9am-5pm, ...
- Examples
  - wd {Sun-Sat}
  - wd {Mon-Fri} hr {9am-4pm}
  - wd {Mon Wed Fri} hr {9am-4pm}, wd{Tue Thu} hr {9am-2pm}
  - sec {0-4 10-14 20-24 30-34 40-44 50-54}

Clients

- "mon" protocol, registered port 2583 with IANA
- Easy Perl interface, Mon::Client
- Get operational status of things monitored
- Disable/enable monitoring and alerting
- Acknowledge alerts sent
- Allows for many reports

Example clients

- **Multiple WWW interfaces**
  - **mon.cgi**
  - **monshow**
  - **minotaur.cgi**
  - **Big Brother facade**
- **Command-line**
- **WAP**
- **2-Way pager**
- **"dtquery" query tool and report generator**

Simple Configuration Example

Send email when any web servers become unpingable:

```
hostgroup webservers www1 www2 www3 www4

watch webservers
    service fping
        monitor fping.monitor
        interval 1m
        period wd {Sun-Sat}
            alert mail.alert trockij
            alertevery 24h
            upalert mail.alert trockij
```

Complex Example

```
watch webserver.corp.com
    service fping
        monitor fping.monitor
        interval 1m
        period P1: wd {Sun-Sat}
            alert mail.alert trockij
            alertevery 12h
            upalert mail.alert trockij
        period P2: wd {Sun-Sat}
            alert mail.alert trockij-pager
            alertevery 24h
            alertafter 3 10m
        period P3: wd {Mon-Fri} hr {7am-10pm}
            alert mail.alert daytime-staff
            alertevery 4h
    service http
        monitor http.monitor
        interval 2m
        depend SELF::fping
        period wd {Sun-Sat}
            alert mail.alert
            alertafter 10m
            numalerts 1
```

Escalation using Multiple Periods

```
watch webserver.corp.com
    service fping
        monitor fping.monitor
        interval 1m
        period P1: wd {Sun-Sat}
            alert mail.alert trockij
            alertafter 3
            numalerts 1
        period P2: wd {Sun-Sat}
            alert qpage.alert trockij
            alertafter 6
            numalerts 1
        period P3: wd {Sun-Sat}
            alert call911.alert
            alertafter 12h
            alertevery 24h
```

Making Monitors

- **Monitors are simple**
  - **expect a list of items to poll from @ARGV**
  - **some standard env variables are set MON_LOGDIR, etc.**
  - **perform tests on items**
  - **first line of output is the summary line**
  - **remaining lines are the detail (not interpreted)**
  - **exit status of zero / nonzero**

Example Monitor

Detect non-operational mountd on NFS servers:

```
#!/usr/bin/perl

my @failed;
my $detail;

foreach my $item (@ARGV) {
    my $output = ‘showmount -e $item 2>&1‘;
    if ($?) {
        push @failed, $item;
        $detail .= "$item failed:\n$output\n";
    }

    else {
        $detail .= "$item ok:\n$output\n";
    }
}

print join (" ", @failed), "\n";
print $detail;

@failed == 0 ? exit 0 : exit 1;
```

Making Alerts

- **Alerts are even simpler than monitors**
  - **@ARGV has some options supplied by server**
  - **rest of @ARGV is from the config file**
  - **first line of stdin is summary**
  - **rest is detail**
  - **perform whatever action desired**

Example Alert

Send email:

```
#!/usr/bin/perl

chomp (my $summary = <STDIN>);

my $to = join (",", @ARGV);

open (MAIL, "| /usr/lib/sendmail -oi -t") || die;

print MAIL <<EOF;
From: mon server
To: $to
Subject: ALERT $summary

Something wicked this way comes.
EOF

close (MAIL);
```

Making Clients

Connect to mon server, download operational status, and display all variables associated with group "server" and service "service":

```
#!/usr/bin/perl

use Mon::Client;

my $cl = new Mon::Client ("host" => "mon-bd2");

$cl->connect;

my %s = $cl->list_opstatus;

$cl->disconnect;

foreach my $var (keys %{$s{"server"}->{"service"}})
{
    print "$var=$s{server}->{service}->{$var}\n";
}
```

Parallelization

Parallelization is handled using two methods:

- **Monitors are parallel processes**
  - **Each "service" process runs independently**
  - **Leverages multiprocessing architectures**

- **Monitors should parallelize their own checks**
  - **Minimize serialization delay when checking numbers of entries**
  - **fping.monitor operates asynchronously**
  - **phttp.monitor operates asynchronously**

Interesting Applications 1

Simlpe home-brew failover

- Several web servers
- Each with eth0 admin and eth0:0 virtual addr
- eth0:0 addresses are published as DNS A records
- mon server polls http servers
- On failure, 'failover.alert' sshs to a 2ndary server and ifup's the dead virtual ip on eth0:1

Interesting Applications 2

Adding on-call schedule support

- Alert uses Schedule::Oncall module
- No changes to the server are needed
- Sends mail to the person on call
- Optionally sends alphanumeric page, also
- Now mon supports on-call schedules!

Interesting Applications 3

Debugging WAN

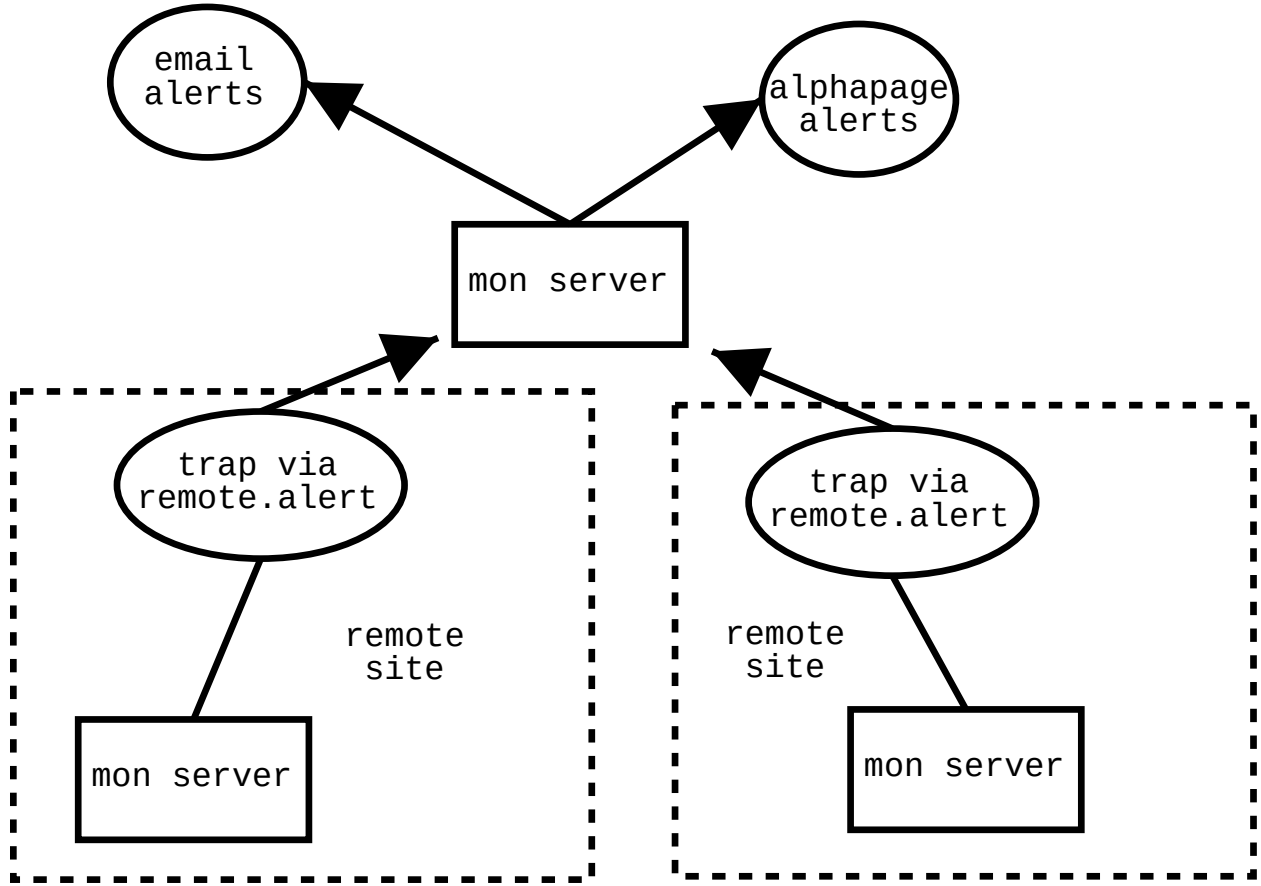
- Traceroute monitor
- Show when path changes
- Record history of traces
- Call ISP with evidence rather than speculation

Interesting Applications 4

Print queues jamming

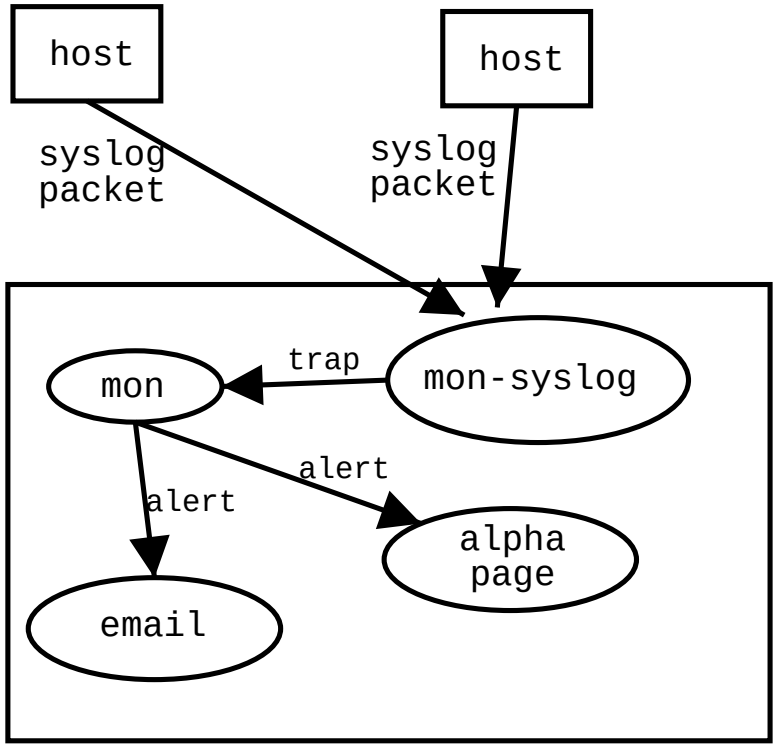
- Clumsy unreliable printers, need to tune lprng
- Catch them when they jam so can collect data
- Shows when a queue is making no progress because of paper or toner deficit

Interesting Applications 5
Hierarchical Monitoring System



Interesting Applications 6

mon-syslog



Interesting Applications 7

dtquery

- **CGI-based tool, mon client**
- **query mon downtime logs for specific downtime events**
- **on specific hosts/groups/services**
- **during specified date ranges**
- **supply with graphs summarizing the results**

Experience

- Useful as a debugging tool
  - Whip-up custom monitors for debugging
  - Logs help investigation of past events
  - Identify that a disaster has been resolved
- If it failed twice before, write a monitor
- Helps keep admins in tune with systems problems
- Admin team knows problems before users report them

Hints

- Take time to tune alerts to maintain your sanity
- Monitor only what you care about, not everything
- That is, keep it simple and digestable
- Use alphanumeric paging via a modem if monitoring networks
- Post your monitors and alerts to the mailing list!

The Design and Operation of "mon"

Jim Trocki
Transmeta Corporation
trockij@transmeta.com

http://www.kernel.org/software/mon/