The Assimilation Monitoring Project

or

How to assimilate a million servers and not get indigestion

#AssimMon

Alan Robertson <alanr@unix.sh>
@OSSAlanR
Project Founder

Primary Project Goals

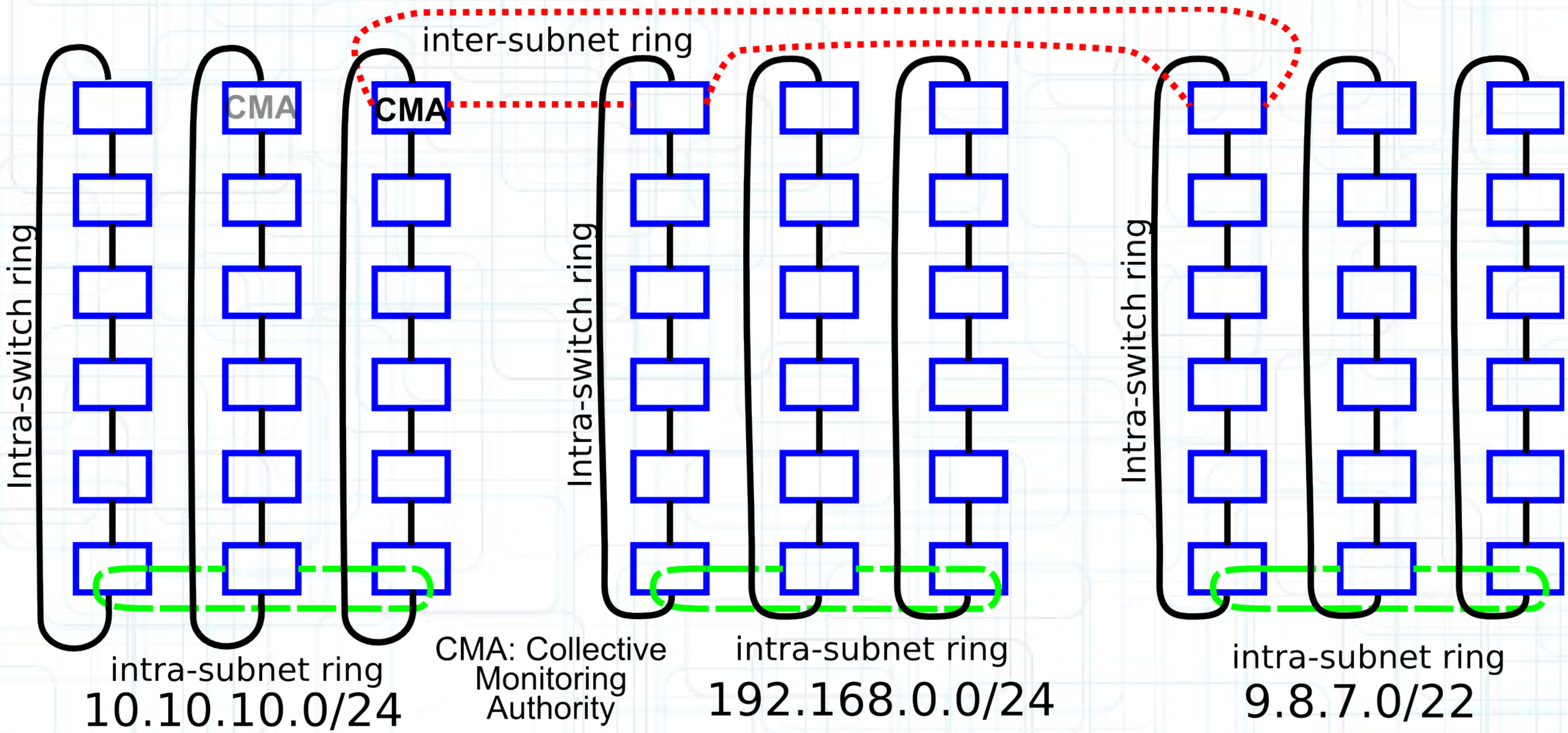
- Exception Monitoring of Systems and Services
- EXTREME monitoring scalability
  >> 10K systems without breathing hard
- Continuous Stealth Discovery of systems, switches, services, and dependencies – *without setting off network alarms*

More about it

- Sub-project of the Linux-HA project
- Personal-time open source project
- Currently licensed under a slightly modified LGPL, small amount GPL
- Currently around 20K lines of code currently
- A work-in-progress

Massive Scalability – *or* “I see dead servers in *O*(1) time”

- **Adding systems does not increase the monitoring work on any system**
- For system monitoring: each talks only to two (or four) neighbors
- Each server monitors its own services
- Ring repair and alerting is $O(n)$ – but a very small amount of work
  - Ring repair for a million nodes is less than 10K packets *per day*



Service Monitoring

- Service monitoring will be performed by the LRM (Local Resource Manager) from the Linux-HA project
- Well-proven code that implements no-news is good news philosophy and supports several classes of resource agents
- Implements the Open Cluster Framework resource agent standard
- LRM is able to start and stop resources (including migrating virtual machines)
- Each system monitors its own services

Monitoring Pros and Cons

Pros

Scalable to virtually any size without rearchitecting central infrastructure

Very simple – easy to understand

Fair, uniform distribution of workload

No single point of failure

Easy to distinguish switch failure vs host failure

Geographically sensitive

Very lightweight on network

Cons

Requires active agents on all machines

More complex than a ping script

Potential slowness at data center power-on

Continuous Stealth Discovery

Continuous - discovers changes in minutes

Stealth - no network packets sent for discovery

Discovery - of what?

- Systems (IP addresses)
- Server-connected switches – and configuration information
- Services – with detailed information
- Service dependencies – who is client of whom
- Network and other configuration

Why continuous discovery?

- Initial configuration is **labor intensive,** current configuration is often out of date.
- Unmonitored servers and services are **slow to repair**
- Knowing the **dependencies** helps track back to the ***root cause*** – reducing time to repair
- Initial configuration quickly becomes **out-of-date**
- Configuration update processes are rarely bulletproof, often **manual**

Why does this matter?

Supports Monitoring

- Helps root cause analysis of cascading failures
- Tells you what you are *not* monitoring
- Helps speed/automate monitoring configuration
- Not confused by non-standard ports
- Closes holes in processes
- Does not trigger network security alarms

System understanding and documentation

- A great many systems are poorly documented, poorly understood
- Creates a rich repository of information about the data center suitable for data mining and auditing
- Could be used to create or audit an ITIL CMDB

Discovering Switches

How?

By listening to one of...

- **CDP** – Cisco Discovery Protocol
- **LLDP** – Link Level Discovery Protocol

Why?

- It helps the $O(1)$ heartbeat protocol
- It tells you how the switches are configured
- It lets you know which systems are plugged into which switch port

Other interesting information gathered

Port number	Port speed
VLAN ID	Port duplex
Port MTU	Switch capabilities

Discovering Systems

How?

Examine the ARP cache

arp -a
cat /proc/net/arp

```
IP address      HW address          Dev
10.10.10.21     6c:62:6d:84:98:a3   eth0
10.10.10.78     00:11:d9:0a:fb:12   eth0
10.10.10.1      20:cf:30:3c:f3:cf   eth0
10.10.10.135    00:04:5a:52:23:75   eth0
10.10.10.4      00:27:13:67:7a:8a   eth0
10.10.10.16     6c:62:6d:84:98:a3   eth0
10.10.10.20     6c:62:6d:84:98:4b   eth0
10.10.10.18     6c:62:6d:84:98:4b   eth0
10.10.10.254    c4:3d:c7:a8:1b:5b   eth0
```

Other information:

- MAC addresses
- Devices

Discovering Services Offered

How?

By examining listening ports

`netstat -tnlp`

```
Local Addr        PID/Program

0.0.0.0:111       763/rpcbind
0.0.0.0:22        667/sshd
0.0.0.0:80        723/apache
0.0.0.0:43935 787/rpc.statd
:::52681          787/rpc.statd
:::111            763/rpcbind
:::22             667/sshd
```

Other Information:

`/proc` contains a wealth of information about the process providing the service

Discovering Service Dependencies

How?

By examining outbound connections

netstat -tnp

comparing to listening ports

Other Information:

/proc contains a wealth of information about the process using the service

netstat -utnp on 10.10.10.5:

```
Foreign Address   PID/Program
10.10.10.4:5900   1639/vncviewer
10.10.10.1:445    - (the kernel)
10.10.10.21:2049 - (the kernel)
```

Listening On 10.10.10.4 (netstat -utnlp)

```
Local Address   PID/Program
:::5900         2265/vino-server
```

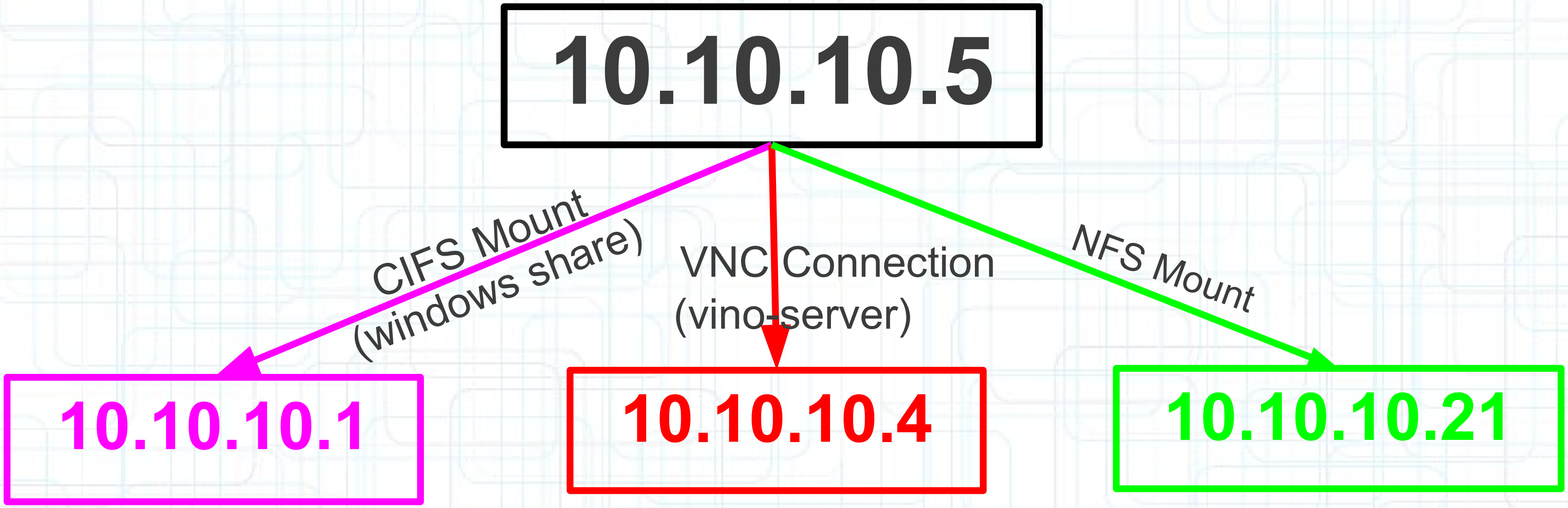
Listening On 10.10.10.21 (netstat -utnlp)

```
Local Address   PID/Program
0.0.0.0:2049     - (the kernel)
```

From /etc/services (a common reference document)

```
nfs            2049/tcp # NFS
microsoft-ds 445/tcp # Microsoft CIFS
```

Service Dependency Graph



Accurate service dependencies greatly assist root cause analysis for cascading failures

Nanoprobe Architecture

Incoming Commands

Monitoring

Server Monitoring

neighbor heartbeats

LRM Proxy

LRM: Local Resource Manager

Discovery

LLDP/CDP

JSON

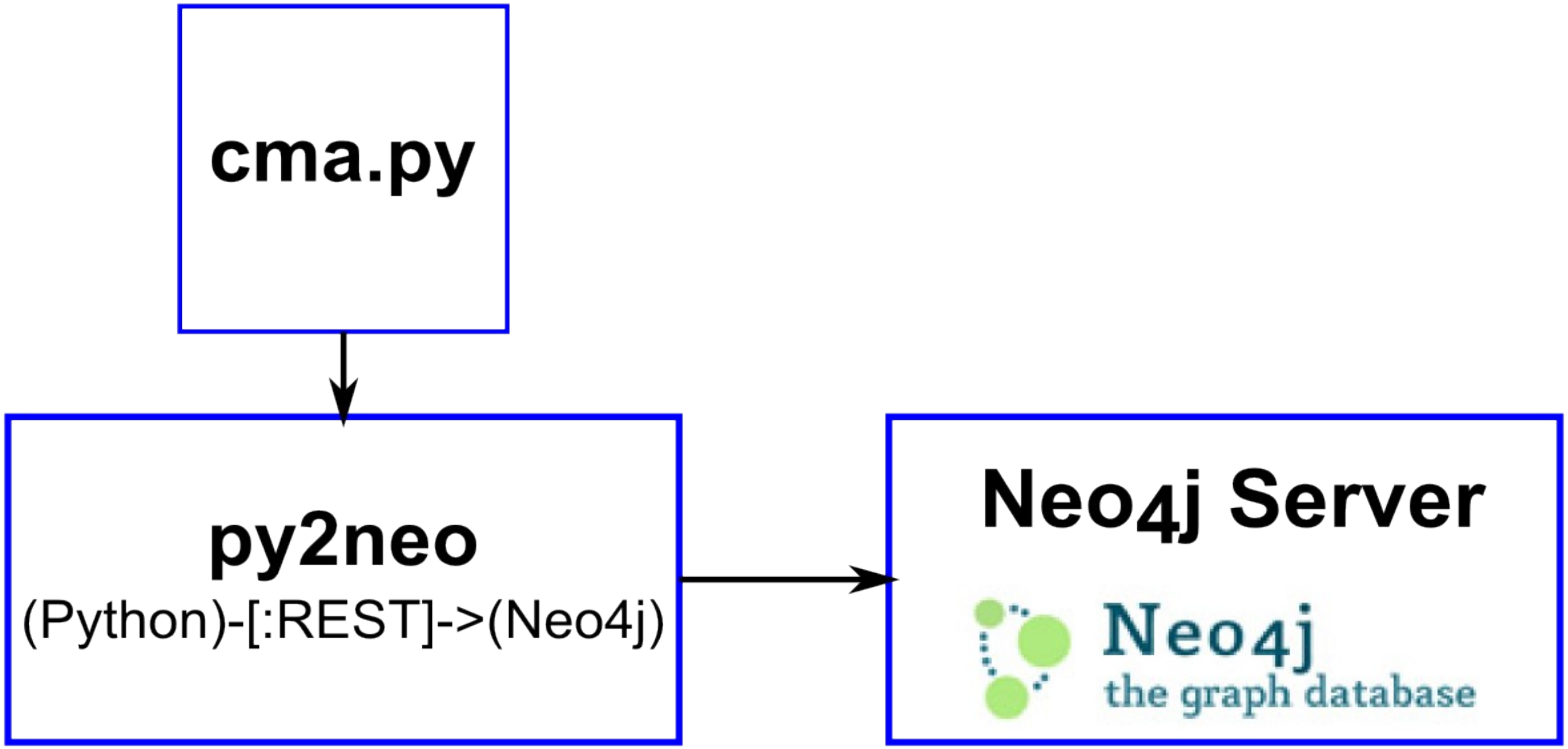
Network
Listening Ports
ARP Cache
etc.

Infrastructure

Scheduling / Marshalling / Demarshalling

Net Address management / Network I/O / Misc

Current (prototype) CMA Architecture



Current Status

- Nanoprobe code up and operational
- Five discovery agents written:
  - CDP/LLDP
  - Network configuration
  - Listening Ports
  - OS discovery
  - CPU discovery
- Initial Collective Management code written
  - Creates a single ring
  - In-memory data only – database code (Neo4j) is in progress
  - Registers and sets up 250K simulated systems in < 11 mins.
  - Currently writing code to construct system graph

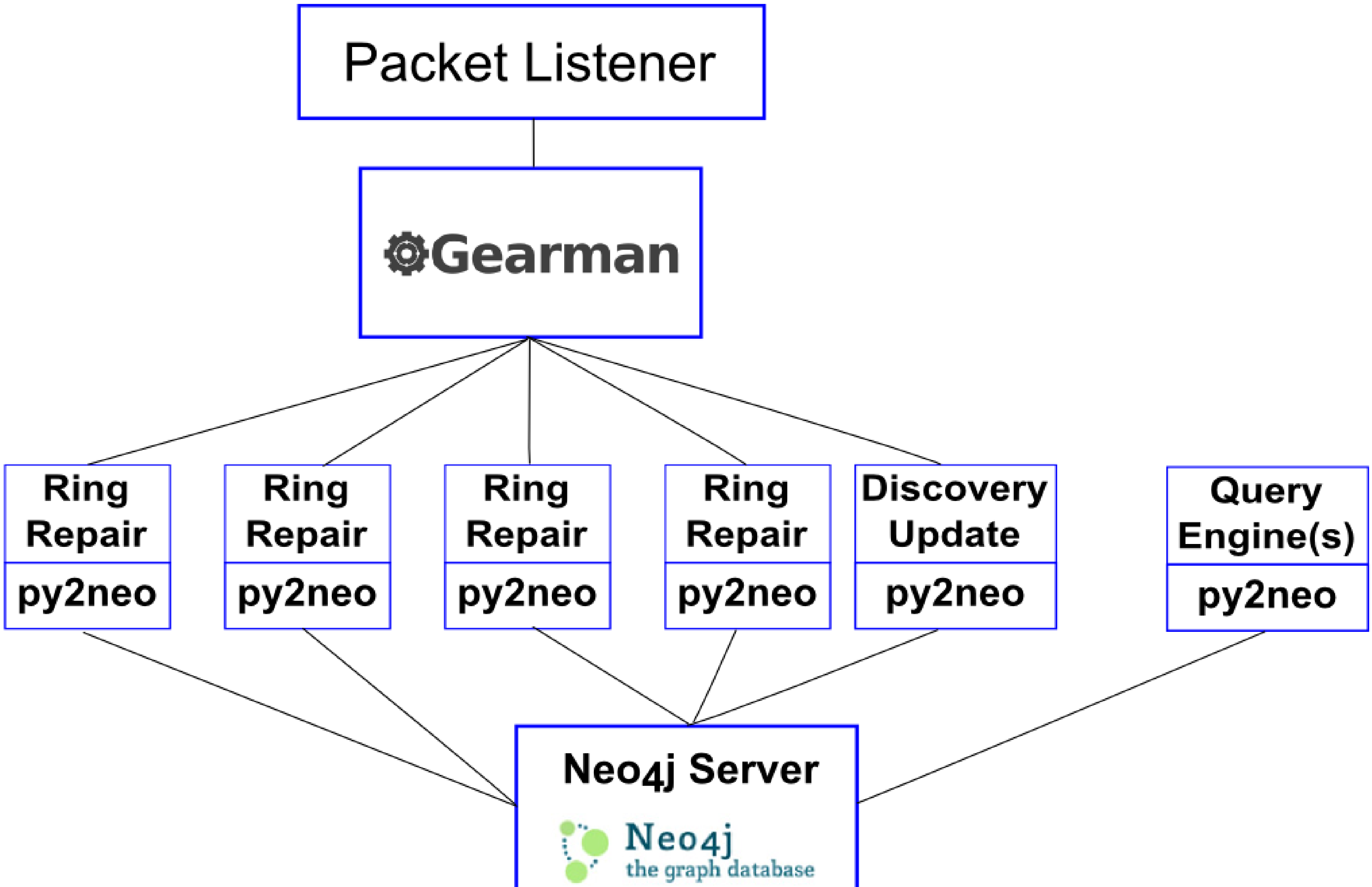
Future Plans and Possibilities - 1

- Neo4j transactions
- Production release (4Q?)
- Clustering for HA and Capacity (see future architecture)
- Linux-HA LRM integration
- Resource control (start/stop/migrate well as monitoring)
- Community building
- Multi-ring architecture
- Appliance monitoring
- Configuration GUI, mobile support, visualization tools
- Importance classification of systems and services

Future Plans and Possibilities - 2

- Multi-tenant support
- Multi-level (physical AND virtual) support
- Role-based access controls
- Auto-monitoring of select classes of services
- More discovery modules (more things and more OSes)
- Limited non-stealth discovery
- Statistical data collection
- Data center analytics (inconsistencies, audits, best practices, diffs, etc)
- Generate ITIL CMDB
- Integration with other systems – trouble ticketing, change management, configuration management, etc.

"Current Thinking" Future CMA Architecture



Net Discovery JSON Snippet


```
"eth0": {
  "address": "00:1b:fc:1b:a8:73",
  "carrier": 1,
  "duplex": "full",
  "mtu": 1500,
  "operstate": "up",
  "speed": 1000,
  "default_gw": true,
  "ipaddrs": {
    "10.10.10.5/24":
         {"brd":"10.10.10.255", "scope":"global", "name":"eth0"}
    "10.10.10.200/24":    {"scope":"global", "name":"eth0:mon"},
    "fe80::21b:fcff:fe1b:a873/64":  {"scope":"link"}
  }
}
```

Listening Port Discovery JSON Snippet


```
"tcp6/::/22": {
    "proto": "tcp6",
    "addr": "::",
    "port": 22,
    "pid": 397,
    "exe": "/usr/sbin/sshd",
    "cmdline": [ "/usr/sbin/sshd", "-D" ],
    "uid": "root",
    "gid": "root",
    "cwd": "/"
}
```