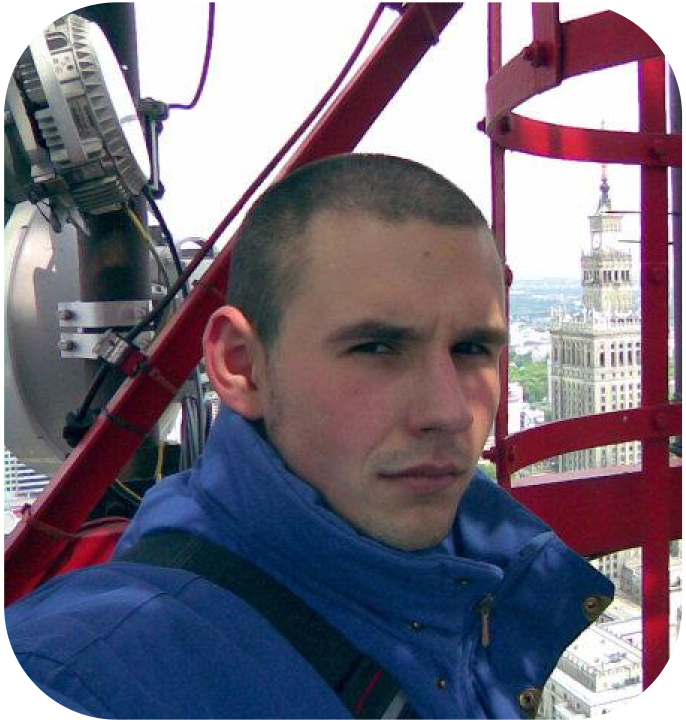
Aggregated Monitoring Chapter One: RouterOS

Bartłomiej Kos
MUM Slovenia
25-26 Feb 2016

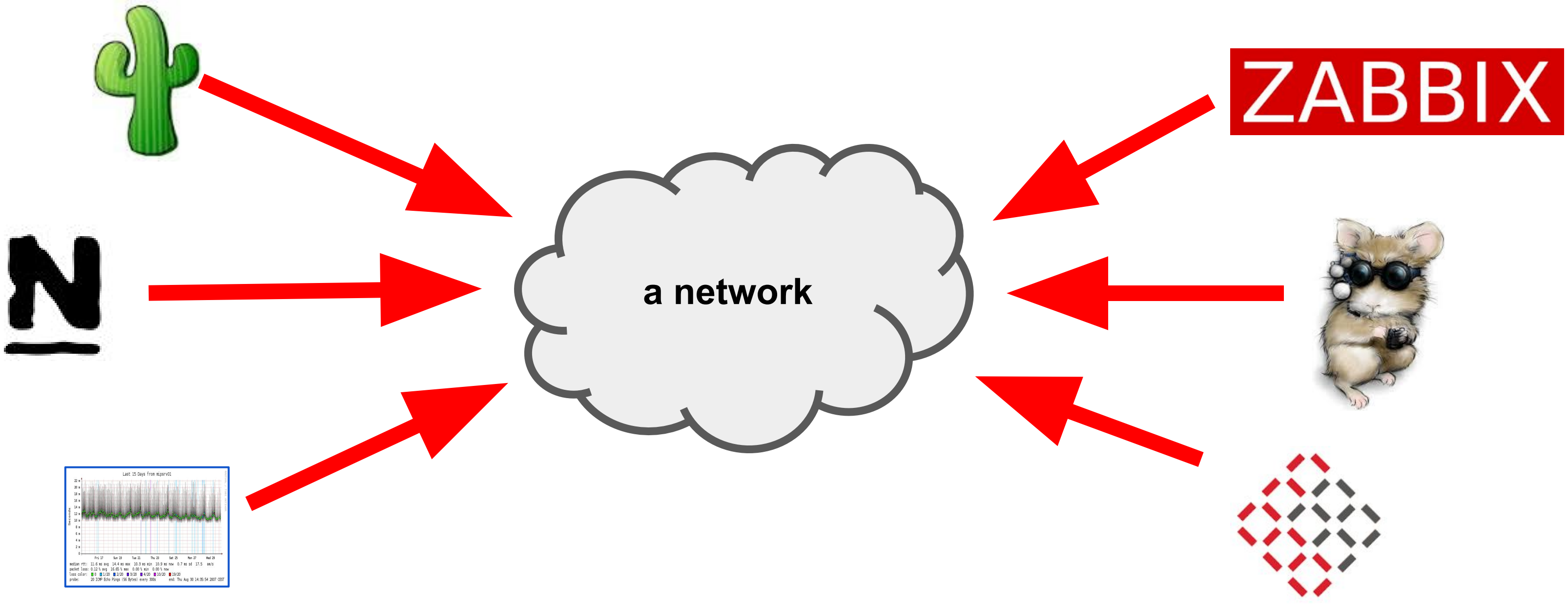
Who am I?

Bartłomiej Kos
bkos@fast-stable-secure.net

- A network engineer
- 9 years in the business
- A MikroTik fan (MTCRE, MTCWE)



Traditional approach to network monitoring



Traditional approach to network monitoring

- Many systems hitting network devices over and over again to get the same data, but to display different results (graphing, threshold monitoring, reachability)
- All the workload inherent to monitoring heaped upon one monitoring system (scalability issues)
- The need for full network reachability between network devices and the monitoring system

- **The need to accommodate oneself to a monitoring system's limited flexibility in order to monitor one's network**

So what do I want from a monitoring system?

- The ability to handle as many different check types as possible (graphing, threshold monitoring, reachability monitoring, etc.)

- Flexibility that allows speedy addition of new monitoring types, both standard (SNMP) and esoteric (CLI)

- Good scalability (up and out)

- Extreme simplicity for allowing one to quickly learn how to get the most of the system without the need to learn some complicated coding techniques (educational as well as operational benefits)

Going Aggregated some two years ago: the base

- Graphing
- Threshold monitoring
- Alerting
- Performance
- Manageability and extensibility

The choice:

Nagios® + PNP4Nagios

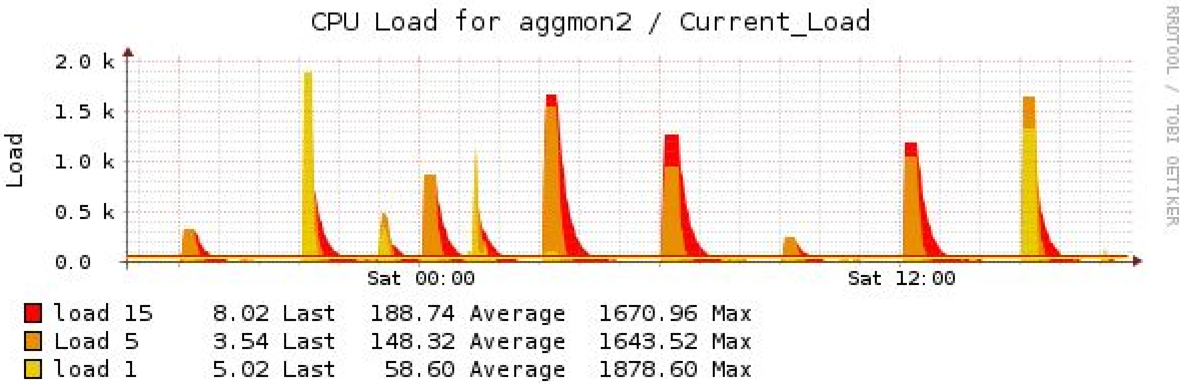
Going Aggregated some two years ago: the architecture

And so, the Aggmon was born...

- Graphing: pnp4nagios
- Threshold monitoring: Nagios Core
- Alerting: Nagios Core
- Extensibility:
  - check_snmp as the base for most of the sensors
  - custom Bash scripts for handling more esoteric checks
  - even more custom Bash scripts for extracting and processing console retrieved data
  - Grand Unified Gatherer Bash script as the means of discovering and identifying devices, and adding and updating their respective sensor configurations
- Performance: good at first, but then, well...

Going Aggregated some half a year ago: the performance

# Active Host / Service Checks: 1067 / 20087



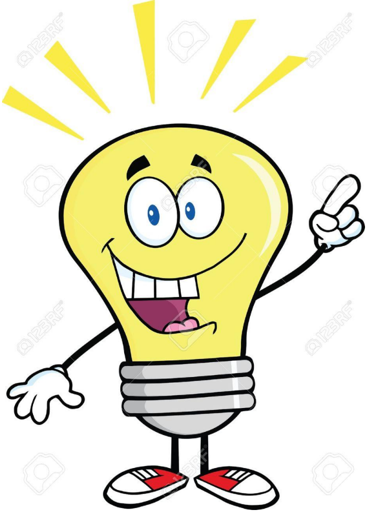
```
🟥 load 15     8.02 Last   188.74 Average   1670.96 Max
🟧 Load 5      3.54 Last   148.32 Average   1643.52 Max
🟨 load 1      5.02 Last    58.60 Average   1878.60 Max
```

Going Aggregated some four months ago: the big break

bkos: *Hey, I’m developing this really great monitoring system thing; can I snmpwalk and snmpget some of your boxes?*

psi: *No, you can’t. But you can use a statistics broker that’s available with my boxes’ central management system.*

bkos: *A statistics broker?*



Going Distributed: situational analysis

What a monitoring system usually does?

- **Devices discovery and identification**
- **Sensors setup**
- **Data retrieval**
- **Data processing**
- **Data presentation**
- **Historical data storing**

Going Distributed: the Aggmon condition

devices discovery,
devices identification,
sensors setup (!!!!!!)

data retrieval (!!!)
and
data processing (!!!)

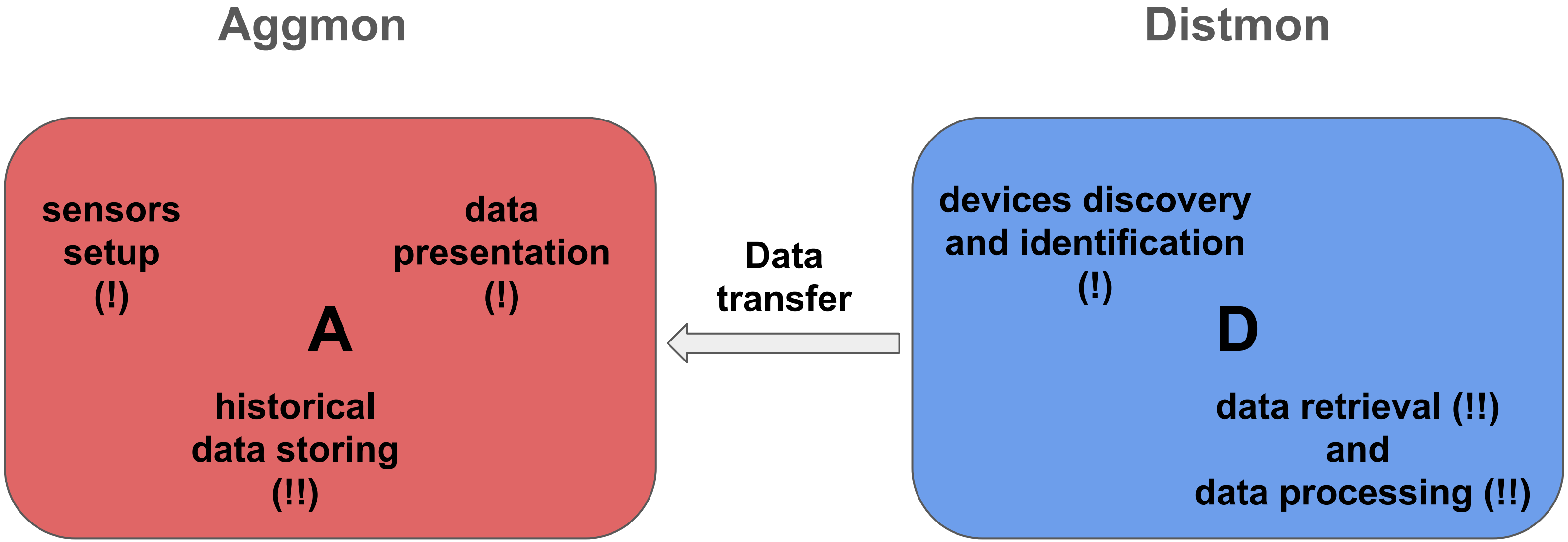
Aggmon

data presentation (!)

historical data storing (!!)

A lot of Compute + A lot of Storage

Going Distributed: the Distmon condition



Aggmon

Distmon

A

sensors setup (!)

data presentation (!)

historical data storing (!!)

Data transfer

D

devices discovery and identification (!)

data retrieval (!!) and data processing (!!)

A lot of Storage

A lot of Compute

Distmon: the cookbook

Codingware required

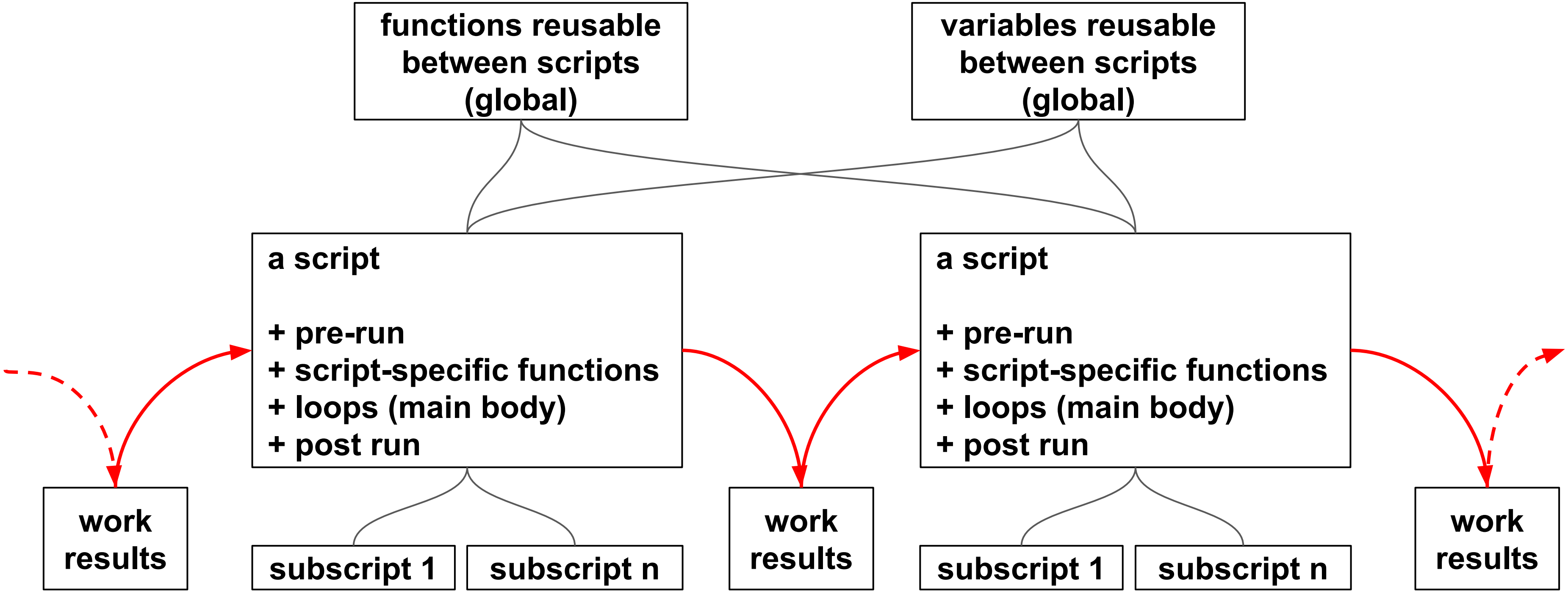
- A Linux (a BSD can do but needs extra care as far as compatibility goes)
- The GNU Bourne Again SHell (Bash)
- grep, sed, and some other standard GNU utilities
- ping and net-snmp
- OpenSSH (ssh, scp)
- GNU bc (for big numbers and floating point calculations)

- Github

Distmon: the recipes

- Devices discovery and identification
    - distmon_runner.sh
- Data retrieval
    - distmon_rover.sh
- Data processing
    - distmon_regulator.sh
    - distmon_presenter.sh
- Sensors setup
    - aggmon_gatherer.sh
- Data presentation and historical data storing
    - Nagios Core + pnp4nagios

Distmon: a simplified block diagram of sorts



functions reusable between scripts (global)

variables reusable between scripts (global)

a script

+ pre-run
+ script-specific functions
+ loops (main body)
+ post run

a script

+ pre-run
+ script-specific functions
+ loops (main body)
+ post run

work results

subscript 1

subscript n

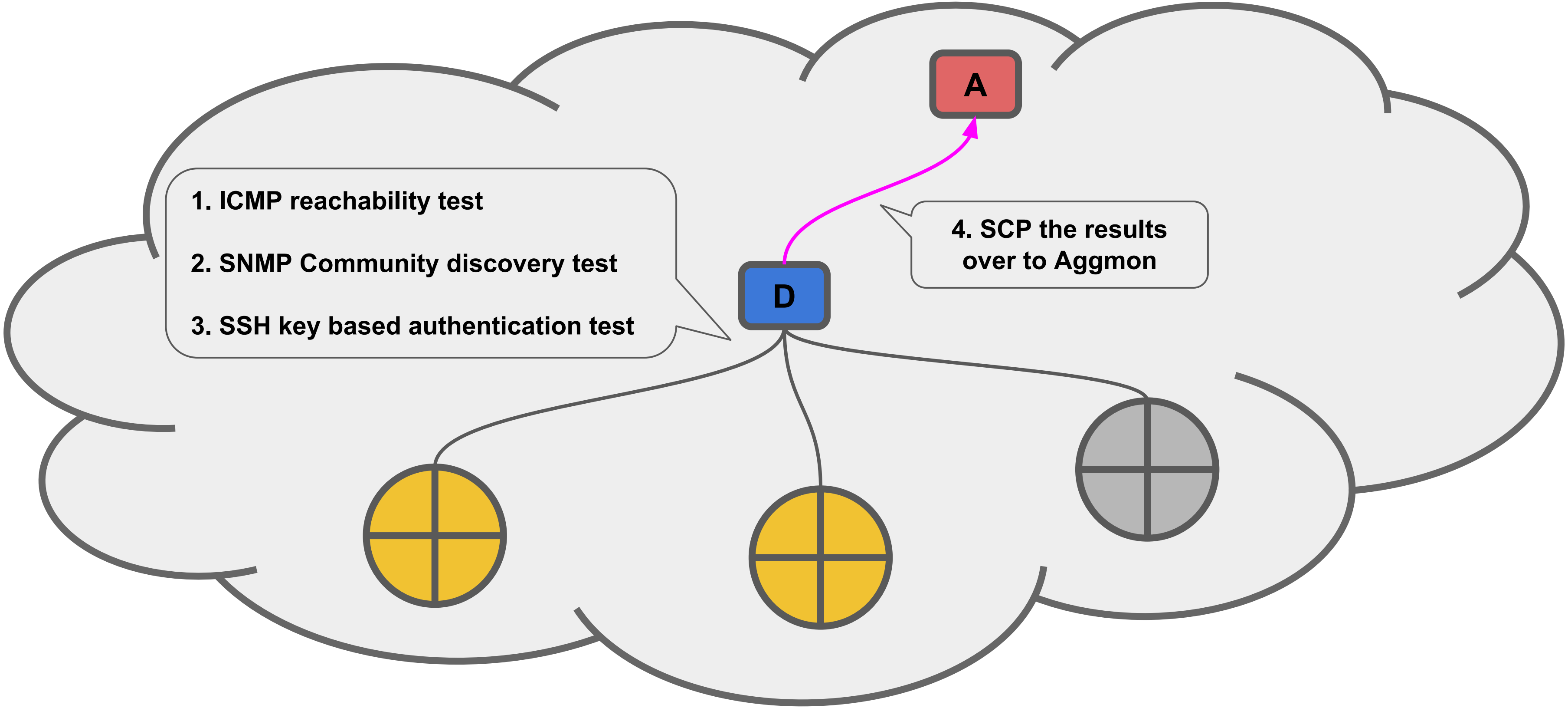
work results

subscript 1

subscript n

work results

Distmon Runner



1. ICMP reachability test

2. SNMP Community discovery test

3. SSH key based authentication test

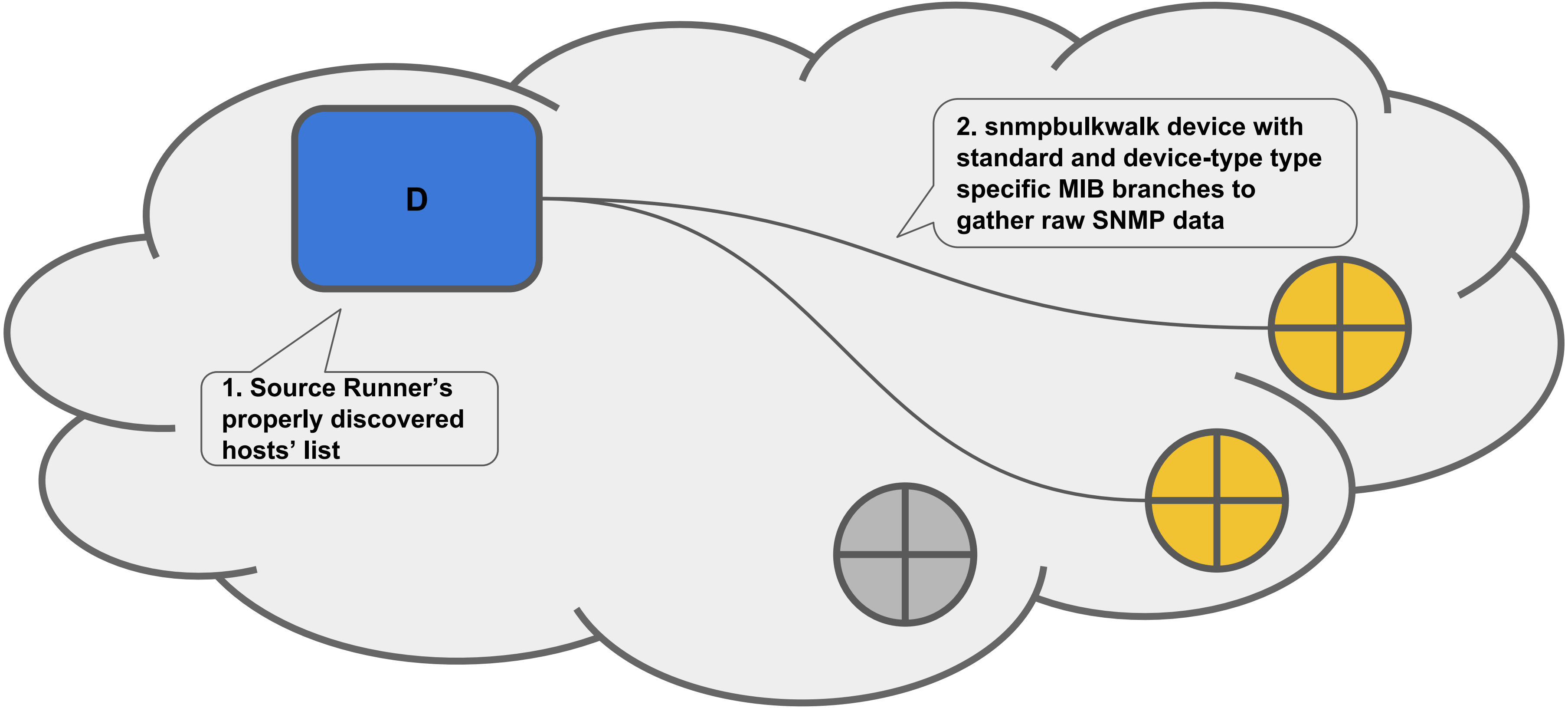
4. SCP the results over to Aggmon

Distmon Runner

- Simple checks
  - Is the host reachable?
  - Does the host respond to a known SNMP Community string?
  - If it does, get its SNMP sysDescr.0
  - Also try to find if the device accepts key-based passwordless login
  - Hosts without a matching SNMP Community or unreachable will not get processed by the other scripts until Runner properly discovers them

- The gathered data stays with the Distmon server, a copy is sent over to Aggmon server(s) via SCP

Distmon Rover



2. snmpbulkwalk device with standard and device-type type specific MIB branches to gather raw SNMP data

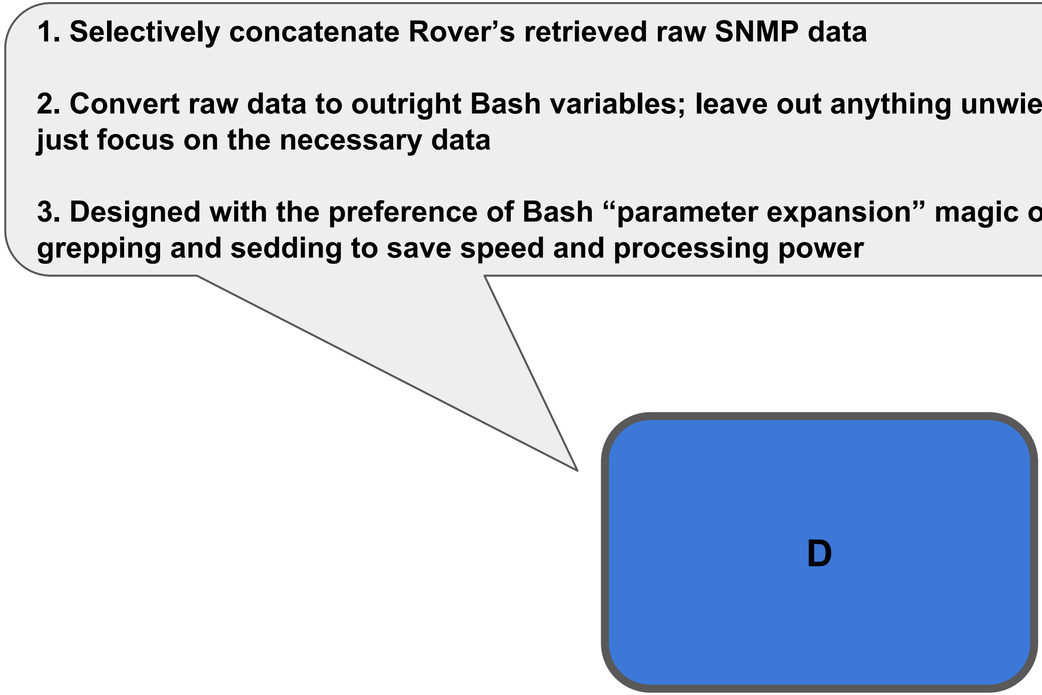
1. Source Runner’s properly discovered hosts’ list

Distmon Rover

- Purpose driven
  - See if the host is up. If it is not, leave it alone until a next scheduled Rover session
  - snmpbulkwalk a host with some generic SNMP MIB branches (IF-MIB::ifTable, etc.)
  - snmpbulkwalk the host with device-specific SNMP MIB branches, depending on device type (MIKROTIK-MIB::mtxrQueues, etc.)
  - Data is saved as retrieved, a timestamp is added for further data processing

- Easy to adapt for data retrieval from new device types (it is just a simple Bash script, after all)
  - if DeviceType1 then GenericMIB + MIB1::branch1 + MIB2::branch2
    if DeviceType2 then DenericMIB + MIB1::branch10 + MIB10::branch1

Distmon Regulator



1. Selectively concatenate Rover’s retrieved raw SNMP data

2. Convert raw data to outright Bash variables; leave out anything unwieldy - just focus on the necessary data

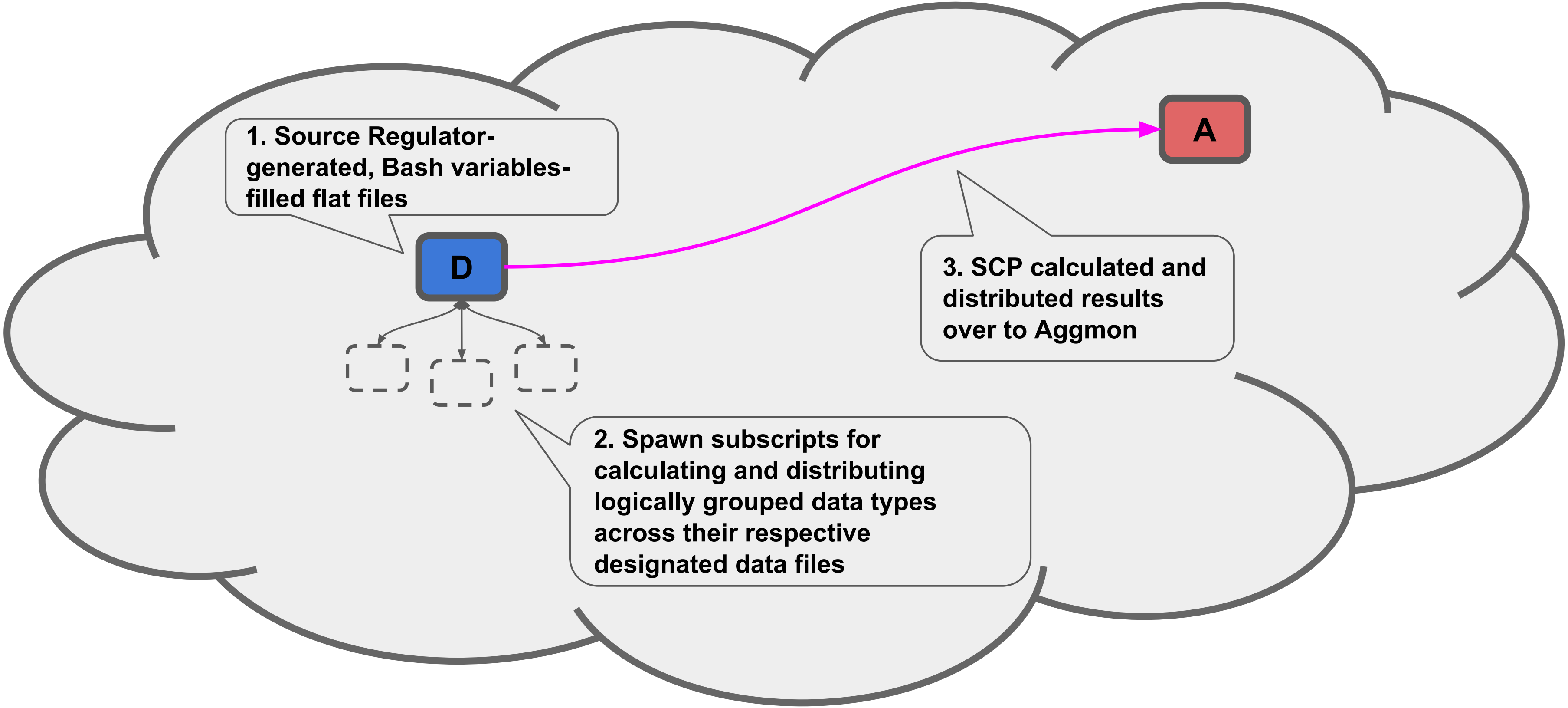
3. Designed with the preference of Bash “parameter expansion” magic over grepping and sedding to save speed and processing power

Distmon Regulator

- The fastest script of the Framework
  - To do its work, it only needs the raw data files gathered by Rover. It does not interact with network devices in any way
  - Regulator makes sure to, in the end, generate files that contain only Bash variables that are ready for sourcing by the next script

- Unless there is a specific need for doing otherwise (and there has not been so far), Regulator takes care of only the data that is easily convertible

Distmon Presenter



1. Source Regulator-generated, Bash variables-filled flat files

2. Spawn subscripts for calculating and distributing logically grouped data types across their respective designated data files

3. SCP calculated and distributed results over to Aggmon

Distmon Presenter

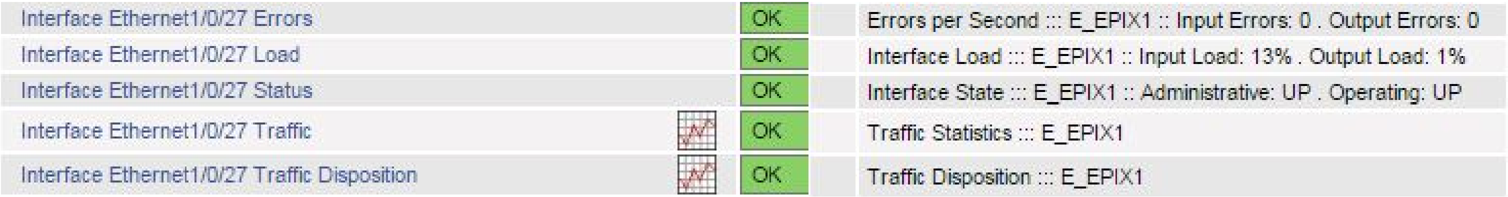
- Responsible for preparing data for direct use by sensors without them having any further need to do any intensive processing of any kind

- Assisted by GNU bc anytime big number (64 bit) or floating point calculations need to be done

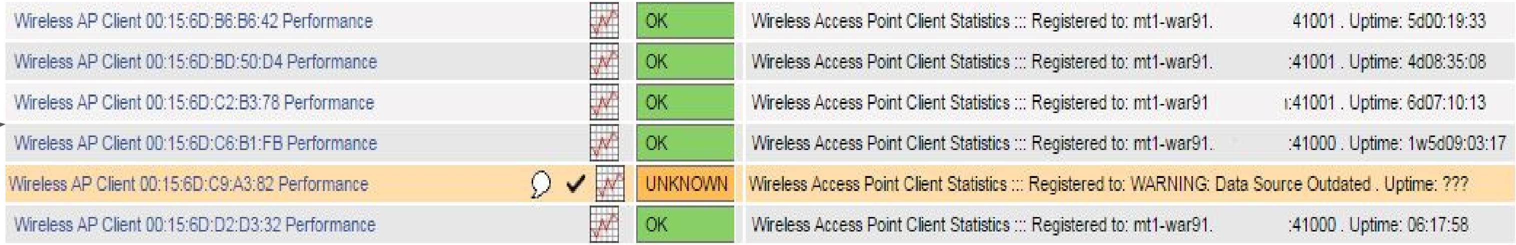
- Relatively the most complicated of all, but easily expandable, owing to subscript-based design (new functionality = new subscript)

- Sends all its calculated data to Aggmon server(s) via SCP

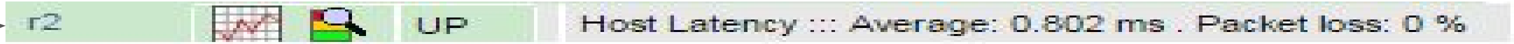
Aggmon Gatherer



Interface Ethernet1/0/27 Errors	OK	Errors per Second ::: E_EPIX1 :: Input Errors: 0 . Output Errors: 0
Interface Ethernet1/0/27 Load	OK	Interface Load ::: E_EPIX1 :: Input Load: 13% . Output Load: 1%
Interface Ethernet1/0/27 Status	OK	Interface State ::: E_EPIX1 :: Administrative: UP . Operating: UP
Interface Ethernet1/0/27 Traffic	OK	Traffic Statistics ::: E_EPIX1
Interface Ethernet1/0/27 Traffic Disposition	OK	Traffic Disposition ::: E_EPIX1



Wireless AP Client 00:15:6D:B6:B6:42 Performance	OK	Wireless Access Point Client Statistics ::: Registered to: mt1-war91. 41001 . Uptime: 5d00:19:33
Wireless AP Client 00:15:6D:BD:50:D4 Performance	OK	Wireless Access Point Client Statistics ::: Registered to: mt1-war91. :41001 . Uptime: 4d08:35:08
Wireless AP Client 00:15:6D:C2:B3:78 Performance	OK	Wireless Access Point Client Statistics ::: Registered to: mt1-war91 ı:41001 . Uptime: 6d07:10:13
Wireless AP Client 00:15:6D:C6:B1:FB Performance	OK	Wireless Access Point Client Statistics ::: Registered to: mt1-war91. :41000 . Uptime: 1w5d09:03:17
Wireless AP Client 00:15:6D:C9:A3:82 Performance ✓	UNKNOWN	Wireless Access Point Client Statistics ::: Registered to: WARNING: Data Source Outdated . Uptime: ???
Wireless AP Client 00:15:6D:D2:D3:32 Performance	OK	Wireless Access Point Client Statistics ::: Registered to: mt1-war91. :41000 . Uptime: 06:17:58



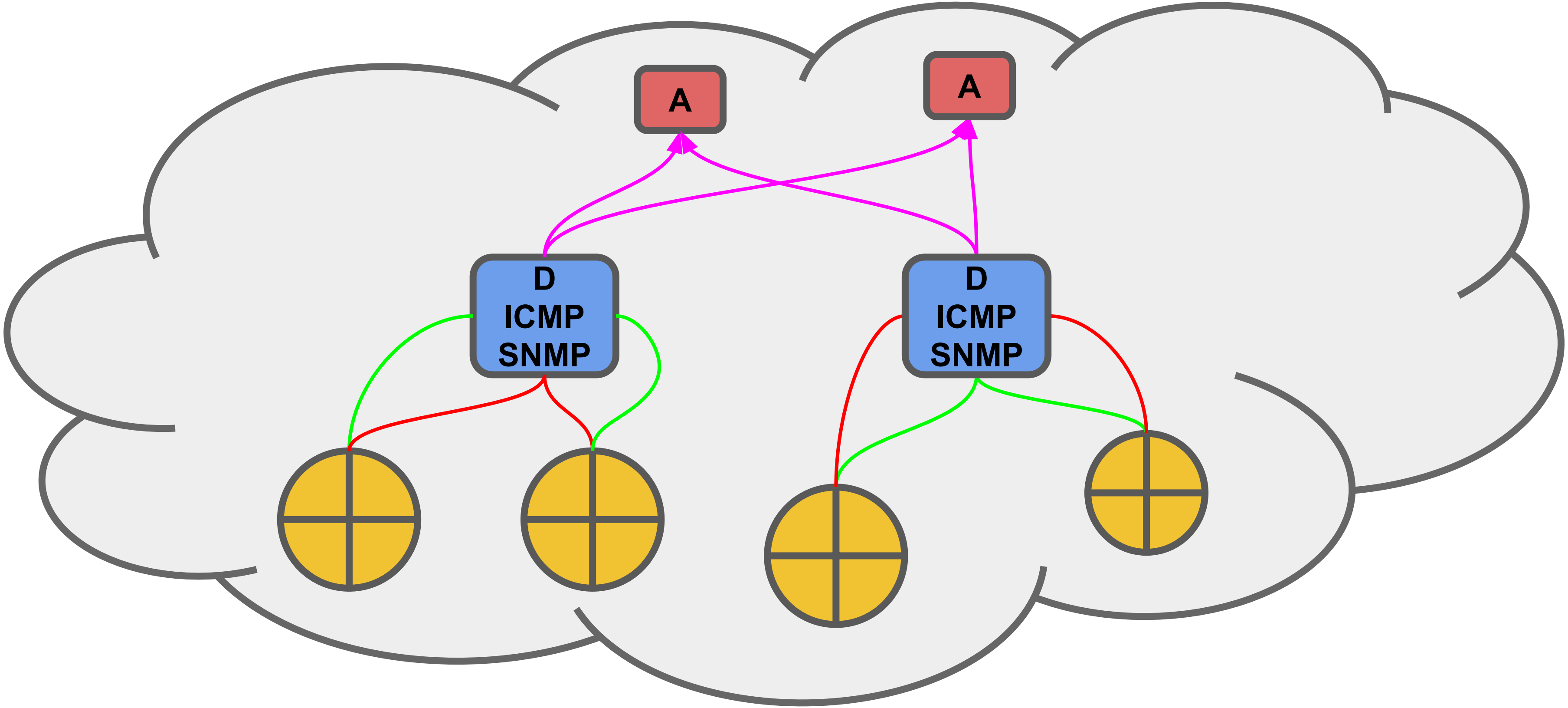
r2	UP	Host Latency ::: Average: 0.802 ms . Packet loss: 0 %

A

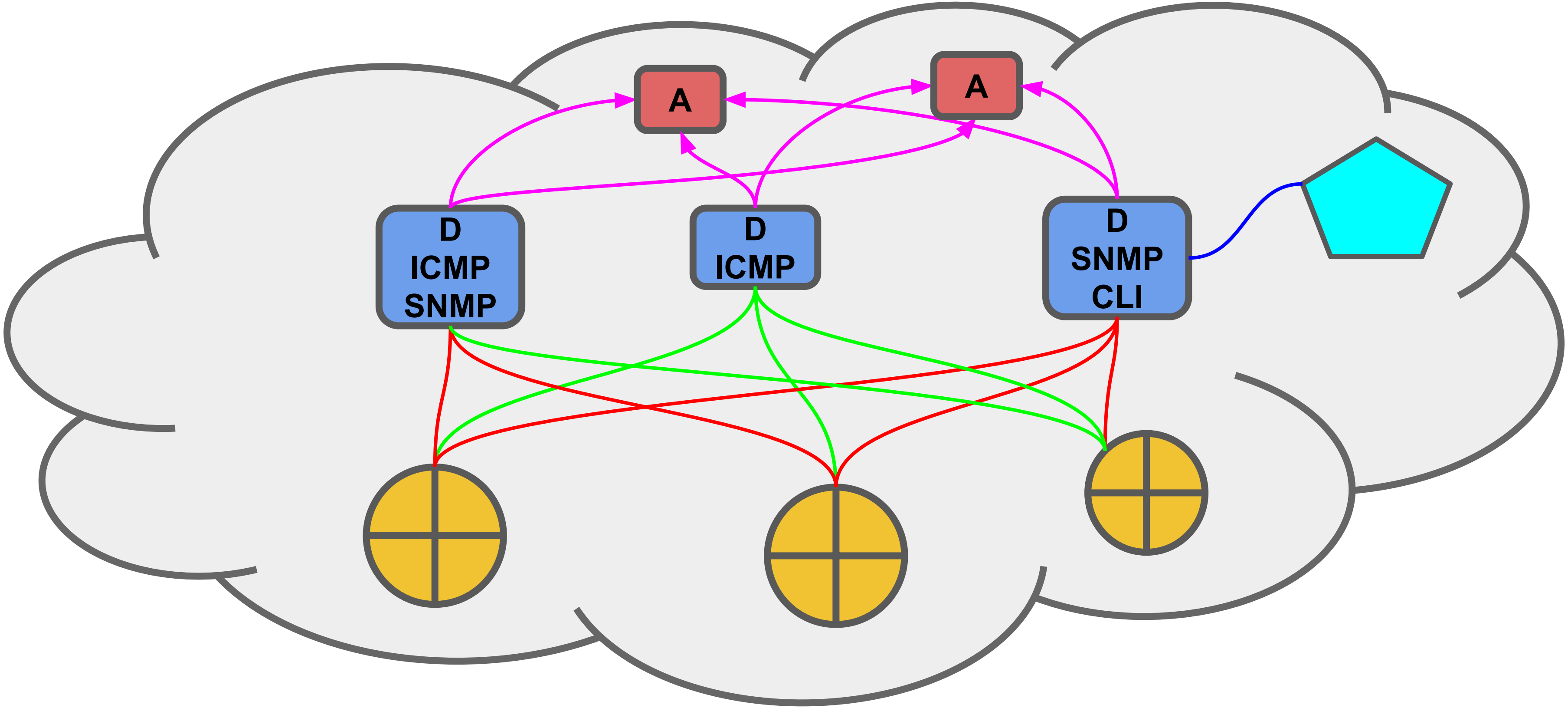
1. **Browse the data received from Distmon(s)**

2. **Generate sensor configurations for running with Nagios Core (threshold monitoring) and pnp4nagios (graphing)**

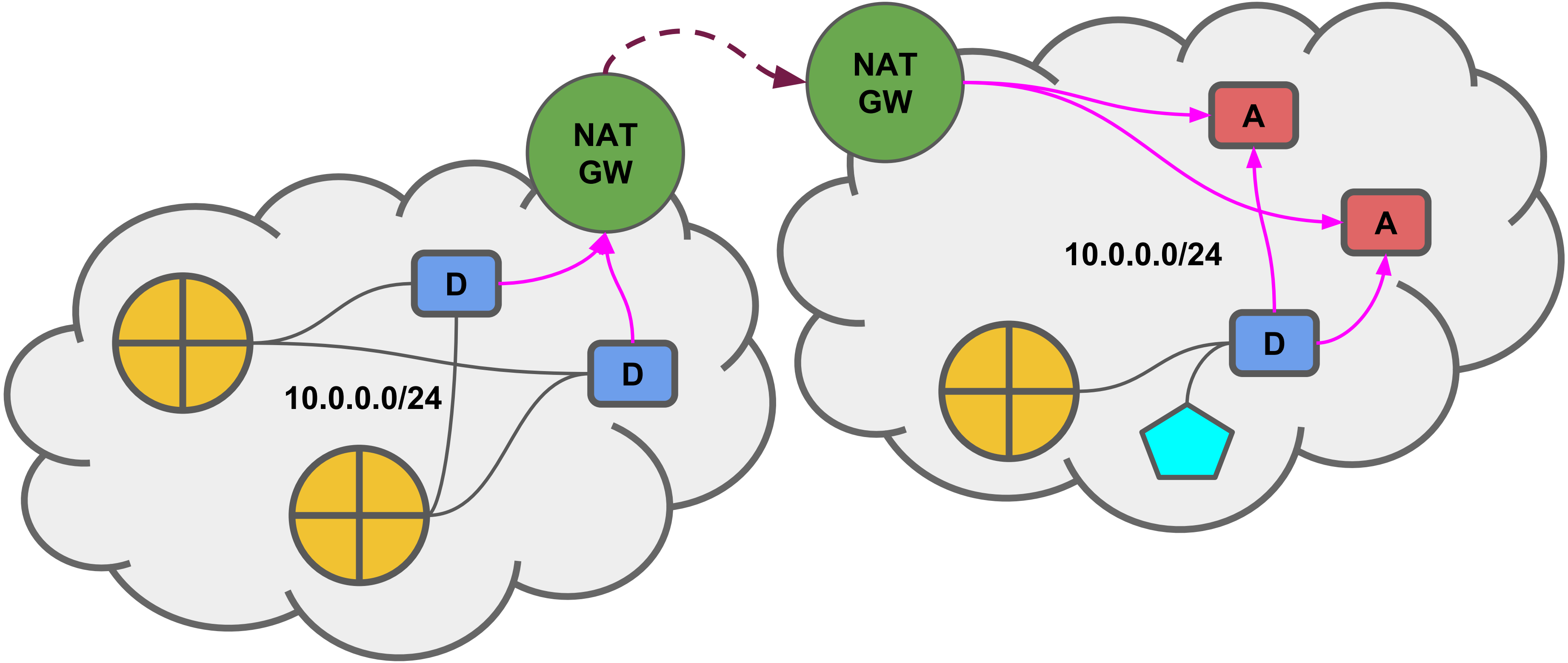
Going Distributed: use case No. 1



Going Distributed: use case No. 2



Going Distributed: use case No. 3



The Aggmon / Distmon performance

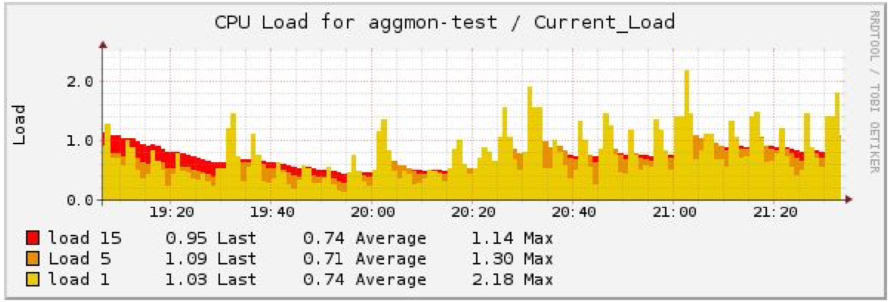
Distmon

```
+ _script_name='Distmon Rover'
+ _script_run_time=11

+ _script_name='Distmon Regulator'
+ _device_count=20
+ _script_run_time=3

+ _script_name='Distmon Presenter'
+ _device_count=20
+ _object_count=1107
+ _script_run_time=8



load average: 1.43, 0.91, 0.75
```

Aggmon

Service Check Execution Time: 0,00 / 0,00 / 0,000 sec
Service Check Latency: 0,00 / 1,00 / 0,016 sec
Host Check Execution Time: 0,00 / 0,00 / 0,000 sec
Host Check Latency: 0,00 / 0,00 / 0,000 sec

# Active Host / Service Checks: 25 / 3393



SNMP interface data

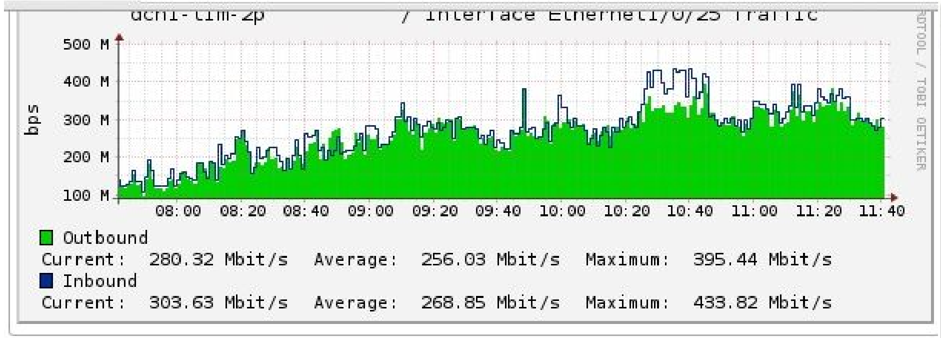
A single object data file provides a wealth of information that can be used by a multitude of sensors

- Traffic graphs (next slide)
- IfUpDown alarm
- IfErrors alarm
- IfUtilisation alarm
- STP transition alarm

And whatever else you can think of putting here!

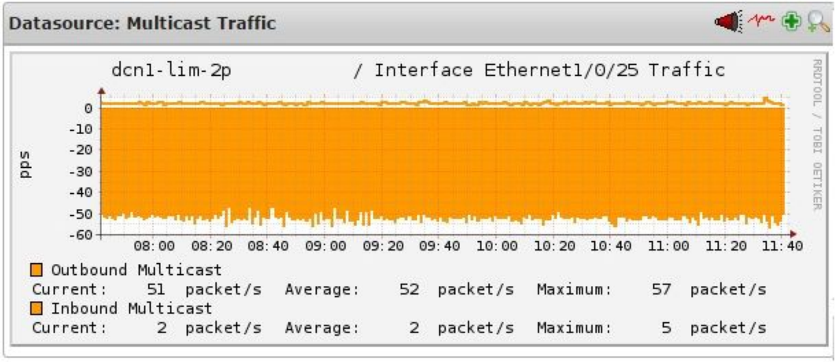
```
ifDescr="Ethernet1/0/27"
ifAlias="E_EPIX1"
ifHighSpeed="10000"
ifAdminStatus="up"
ifOperStatus="up"
inBitsPerSecond="1351290279.20000000000000000000"
outBitsPerSecond="64916717.33333333333333333328"
inUcastPktsPerSecond="6083.01666666666666666666"
outUcastPktsPerSecond="9619.01666666666666666666"
inMcastPktsPerSecond="109693.81666666666666666666"
outMcastPktsPerSecond=".48333333333333333333"
inBcastPktsPerSecond="87.68333333333333333333"
outBcastPktsPerSecond=".05000000000000000000"
relativeInUcastPktsPerSecond="5.25011180443365534100"
relativeOutUcastPktsPerSecond="99.99445573510888399100"
relativeInMcastPktsPerSecond="94.67421072686763029700"
relativeOutMcastPktsPerSecond=".00502449005757372500"
relativeInBcastPktsPerSecond=".07567746869871434600"
relativeOutBcastPktsPerSecond=".00051977483354210900"
inErrorsPerSecond="0"
outErrorsPerSecond="0"
inIfLoad="13.51290279200000000000"
outIfLoad=".64916717333333333333"
stpForwardingStateTransitionsPerInterval="0"
__resultsFreshnessCheckValue="1"
```

SNMP interface graphs



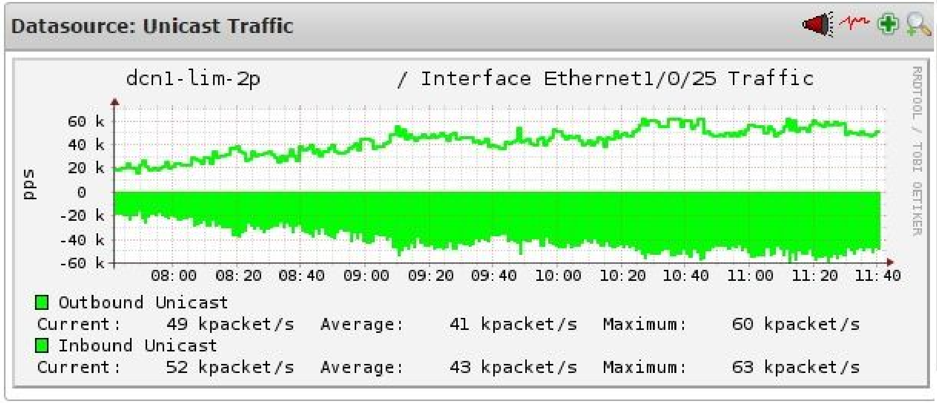
```
🟩 Outbound
Current:  280.32 Mbit/s   Average:  256.03 Mbit/s   Maximum:  395.44 Mbit/s
🟦 Inbound
Current:  303.63 Mbit/s   Average:  268.85 Mbit/s   Maximum:  433.82 Mbit/s
```

Datasource: Multicast Traffic



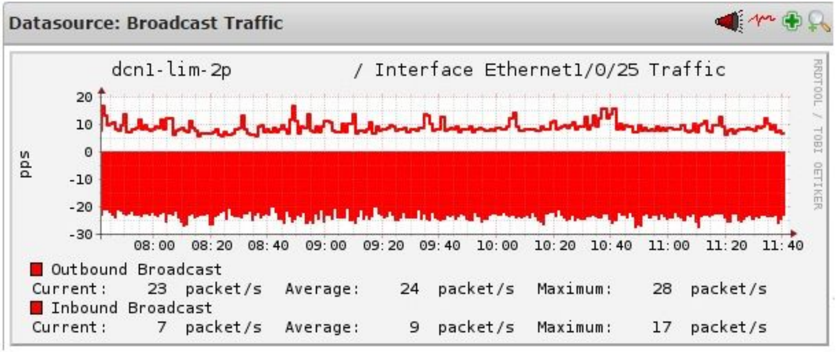
```
🟧 Outbound Multicast
Current:    51  packet/s   Average:    52  packet/s   Maximum:    57  packet/s
🟧 Inbound Multicast
Current:     2  packet/s   Average:     2  packet/s   Maximum:     5  packet/s
```

Datasource: Unicast Traffic



```
🟩 Outbound Unicast
Current:    49 kpacket/s   Average:    41 kpacket/s   Maximum:    60 kpacket/s
🟩 Inbound Unicast
Current:    52 kpacket/s   Average:    43 kpacket/s   Maximum:    63 kpacket/s
```

Datasource: Broadcast Traffic

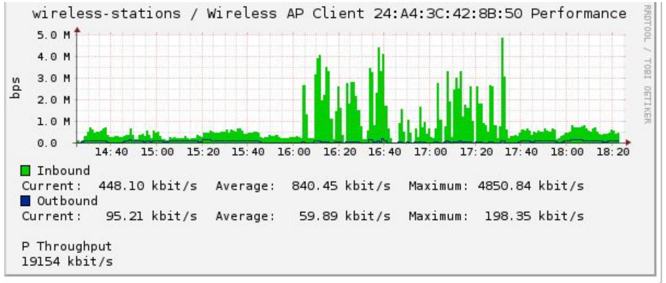


```
🟥 Outbound Broadcast
Current:    23  packet/s   Average:    24  packet/s   Maximum:    28  packet/s
🟥 Inbound Broadcast
Current:     7  packet/s   Average:     9  packet/s   Maximum:    17  packet/s
```

CLI WiFi Registration Table client extracted and parsed

```
id="*1cc";interface="wlan1";radio_name="AirGrid M5 HP";mac_address="24:A4:
3C:42:96:EE";ap="false";wds="false";bridge="false";rx_rate="36.0Mbps";
tx_rate="54.0Mbps";packets="275407,209694";bytes="363721441,23250692";
frames="275407,209785";frame_bytes="364290329,22002264";hw_frames="
336462,212411";hw_frame_bytes="456886638,30538226";tx_frames_timed_out="0";
uptime="07:18:13";last_activity="00:00:02.460";signal_strength="-52
dBm@6Mbps";signal_to_noise="39";signal_strength_ch0="-52";
strength_at_rates="-52dBm@6Mbps 30ms;-52dBm@12Mbps 8m20s130ms;-52dBm@18Mbps
3m33s910ms;-52dBm@24Mbps 3m32s800ms;-53dBm@36Mbps 2s460ms;-52dBm@48Mbps
6s250ms";tx_signal_strength="-48";tx_ccq="77";rx_ccq="99";p_throughput="
29860";distance="1";nstreme="false";framing_mode="none";routeros_version="
2.9.31";last_ip="192.168.192.195";ieee8021x_port_enabled="true";
authentication_type="wpa2-psk";encryption="aes-ccm";group_encryption="aes-
ccm";management_protection="false";compression="false";wmm_enabled="false";
registered_to="mt1-war91:41001"
inBytesPerSecond="6"
outBytesPerSecond="0"
clientUptimeSeconds="26293"
```

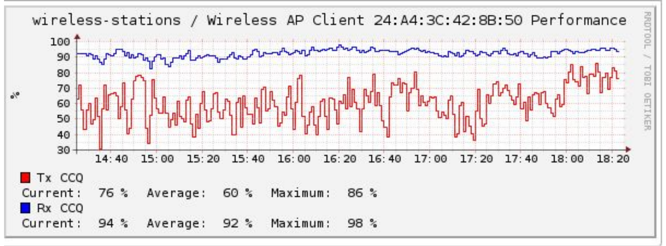
CLI WiFi Registration Table client graphs



🟩 Inbound
Current: 448.10 kbit/s Average: 840.45 kbit/s Maximum: 4850.84 kbit/s
🟦 Outbound
Current: 95.21 kbit/s Average: 59.89 kbit/s Maximum: 198.35 kbit/s

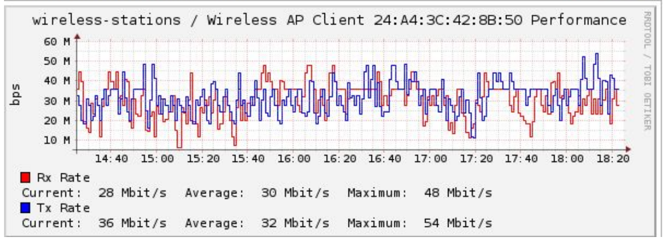
P Throughput
19154 kbit/s

Datasource: outBytesPerSecond

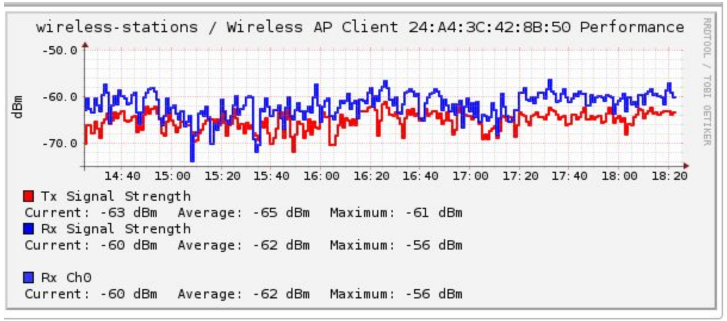


🟥 Tx CCQ
Current: 76 % Average: 60 % Maximum: 86 %
🟦 Rx CCQ
Current: 94 % Average: 92 % Maximum: 98 %

Datasource: clientUptimeSeconds



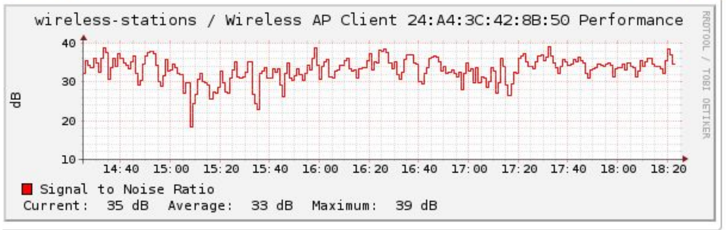
🟥 Rx Rate
Current: 28 Mbit/s Average: 30 Mbit/s Maximum: 48 Mbit/s
🟦 Tx Rate
Current: 36 Mbit/s Average: 32 Mbit/s Maximum: 54 Mbit/s



🟥 Tx Signal Strength
Current: -63 dBm Average: -65 dBm Maximum: -61 dBm
🟦 Rx Signal Strength
Current: -60 dBm Average: -62 dBm Maximum: -56 dBm

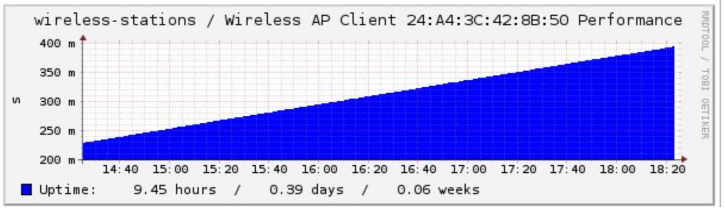
🟦 Rx Ch0
Current: -60 dBm Average: -62 dBm Maximum: -56 dBm

Datasource: txRate



🟥 Signal to Noise Ratio
Current: 35 dB Average: 33 dB Maximum: 39 dB

Datasource: rxSignalStrength



🟦 Uptime: 9.45 hours / 0.39 days / 0.06 weeks

Aggmon and Aggmon / Distmon creation timeframe

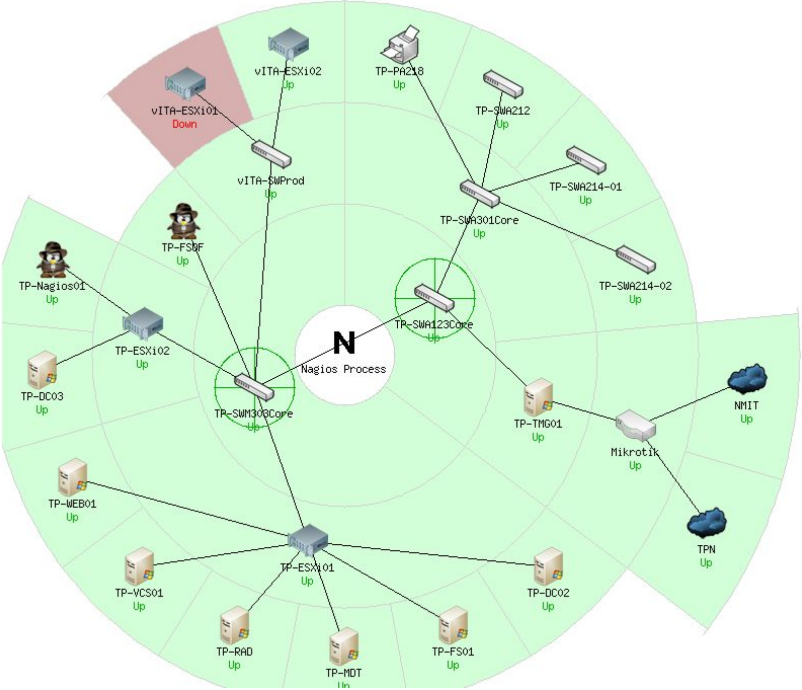
From the initial concept through Aggmon to Aggmon / Distmon as it is available today, it took one guy (myself) about four months of honest labour* in his free time to get the system up and working

I am proud to say that I am now able to efficiently monitor my network to the best possible extent, and am able to introduce new sensors of any kind within several minutes, hours at worst. **I can, but so can you!**

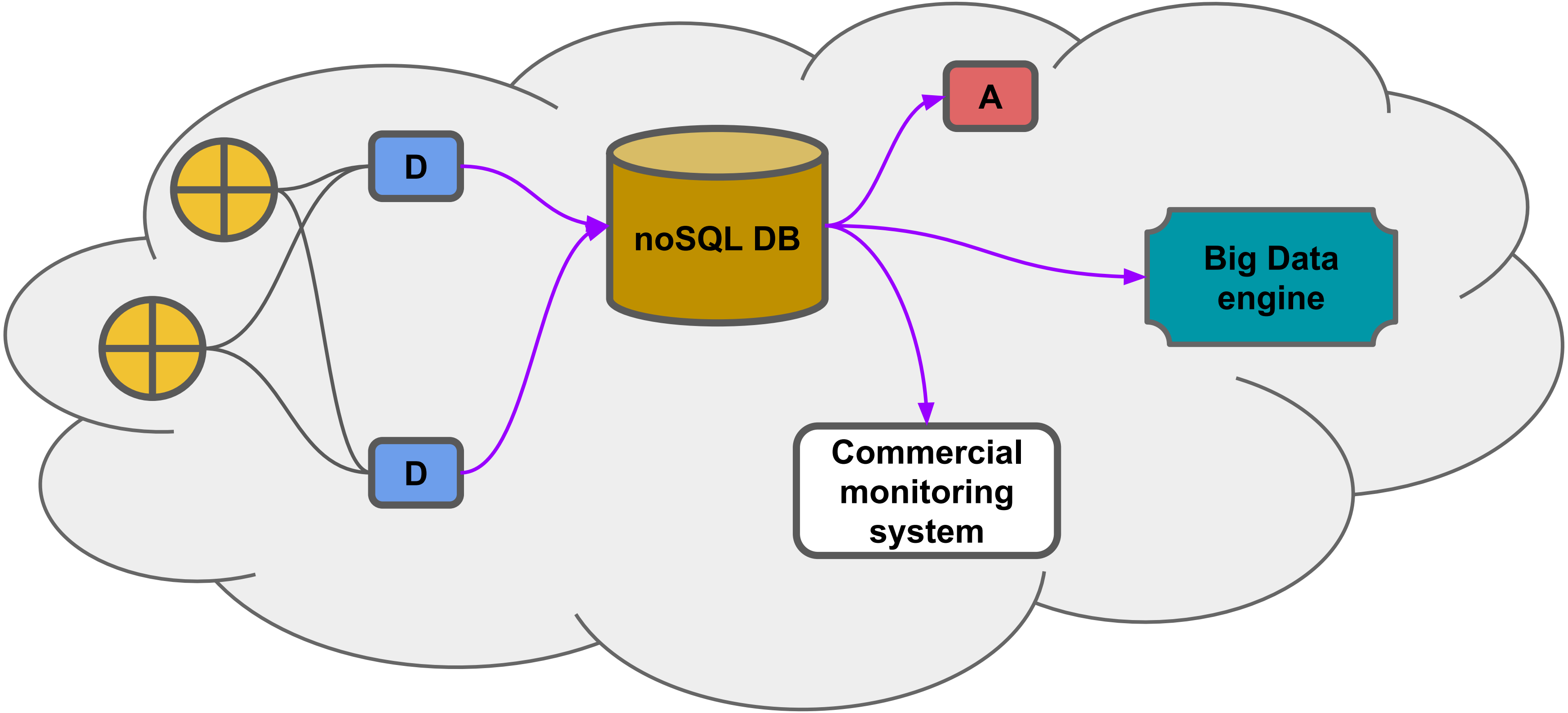
* 1 day of honest labour = 8 hours spent on working on this project only

Plans for the future: Nagios automatic map charting

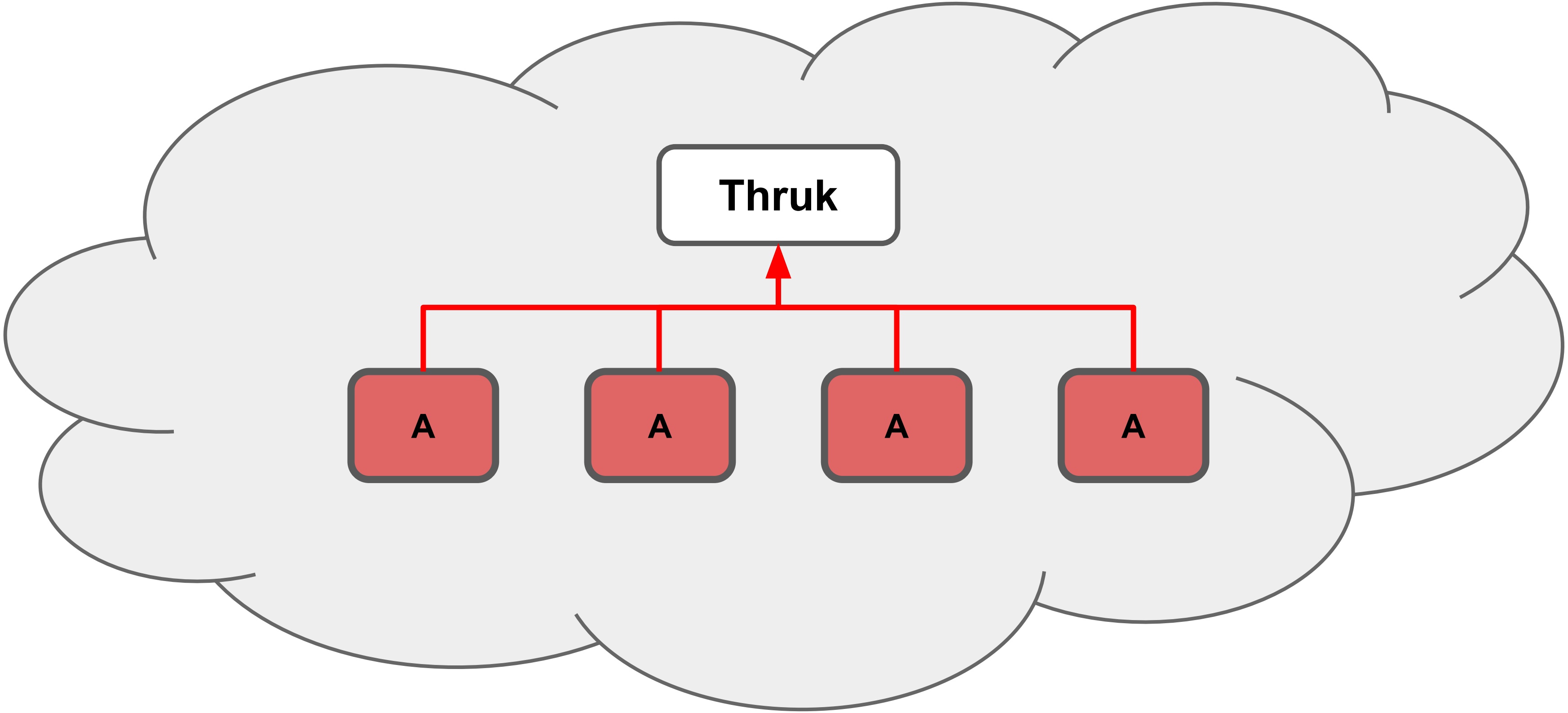
```
ifDescr="Ethernet1/0/27"
ifAlias="E_EPIX1"
...
```



Plans for the future: noSQL database integration



Plans for the future: Thruk ‘scale out’ integration

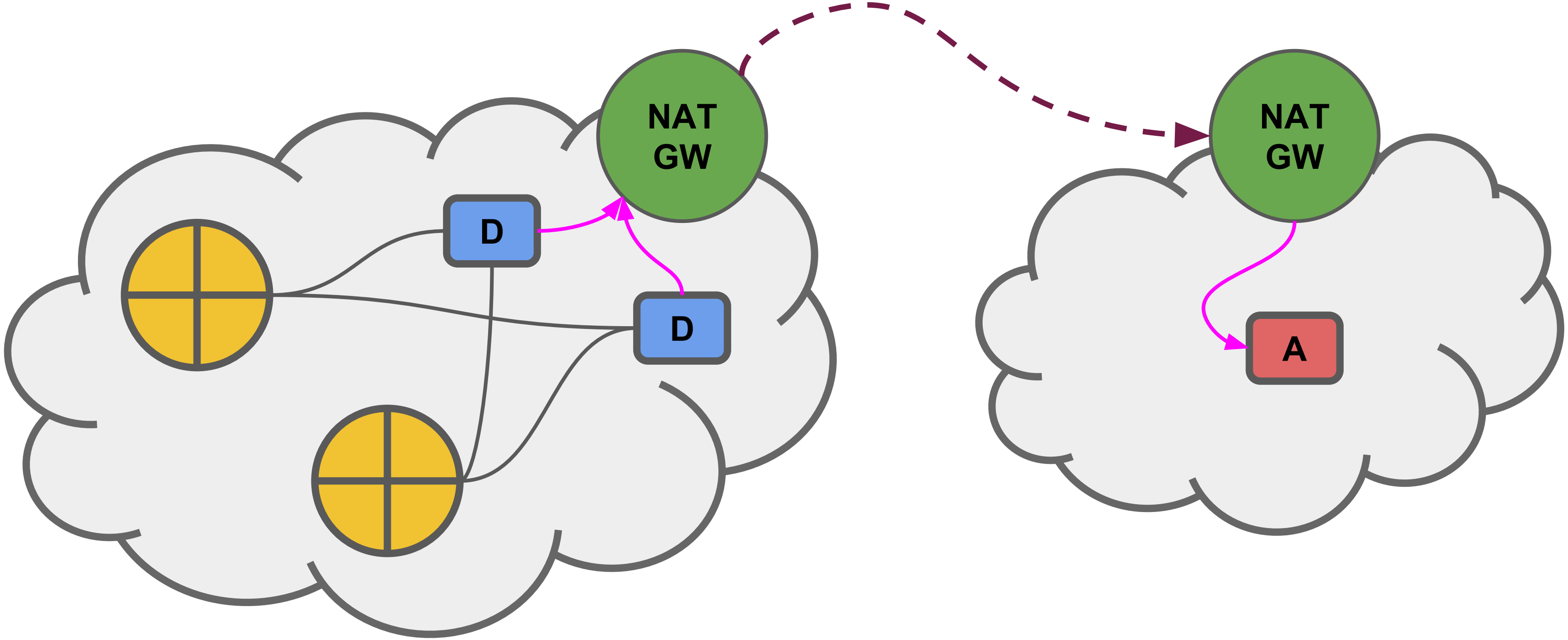


Plans for the future: …

- **Proper developmental documentation!**

- A collaborative effort, perhaps?

Live demo



Thank you for your attention!