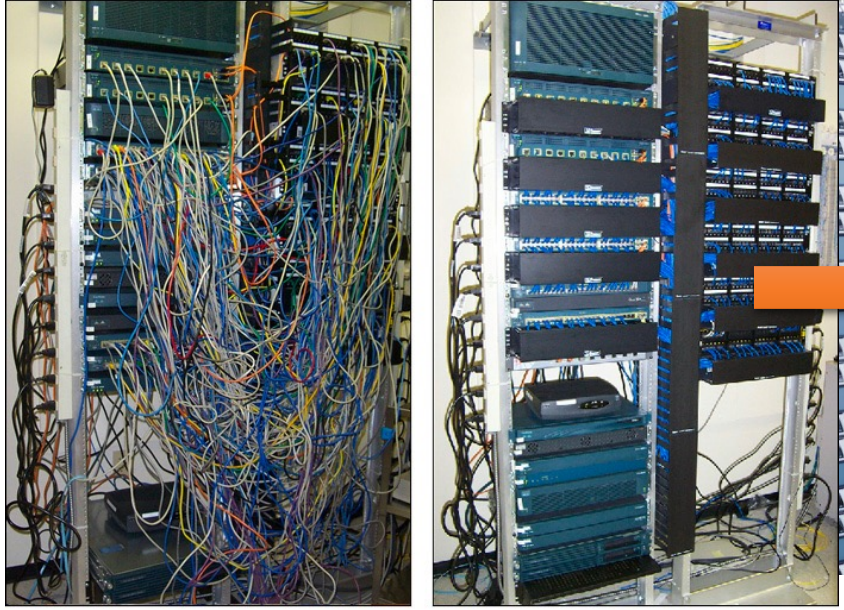
From managing networks to programming networks: Moving beyond the ASN.1 era

Henning Schulzrinne

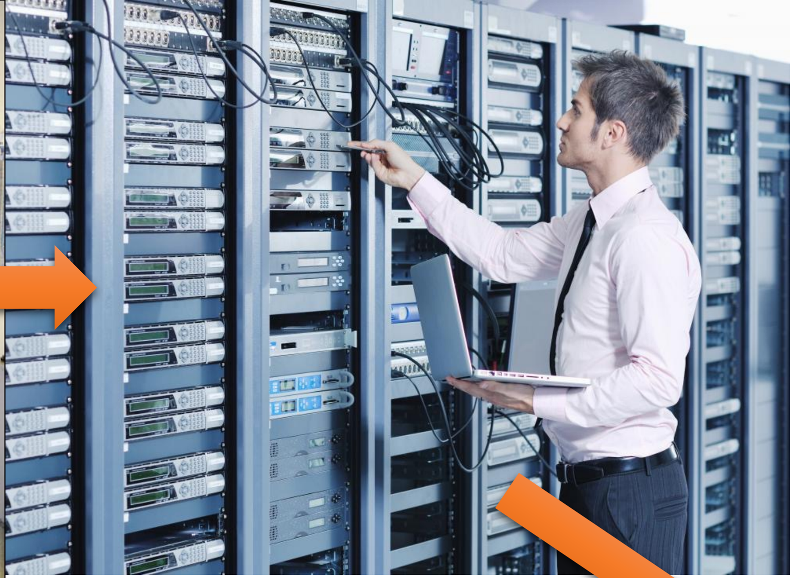
Columbia University

There’s lots of lands that are unknown to me

Managing networks → Programming systems



BEFORE | AFTER



Network management career progression stereotype

network administrator = tech career without programming?

The state of home network management

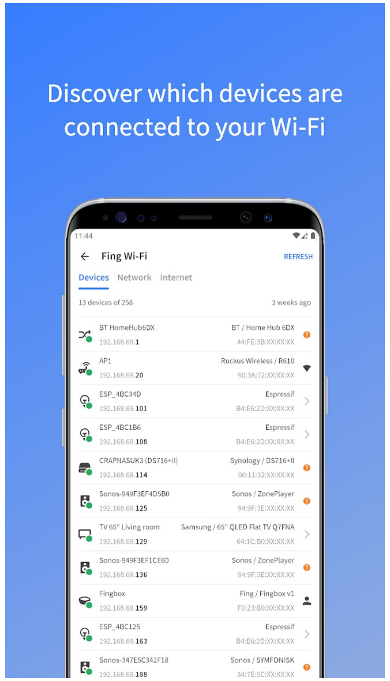
Downdetector verizon

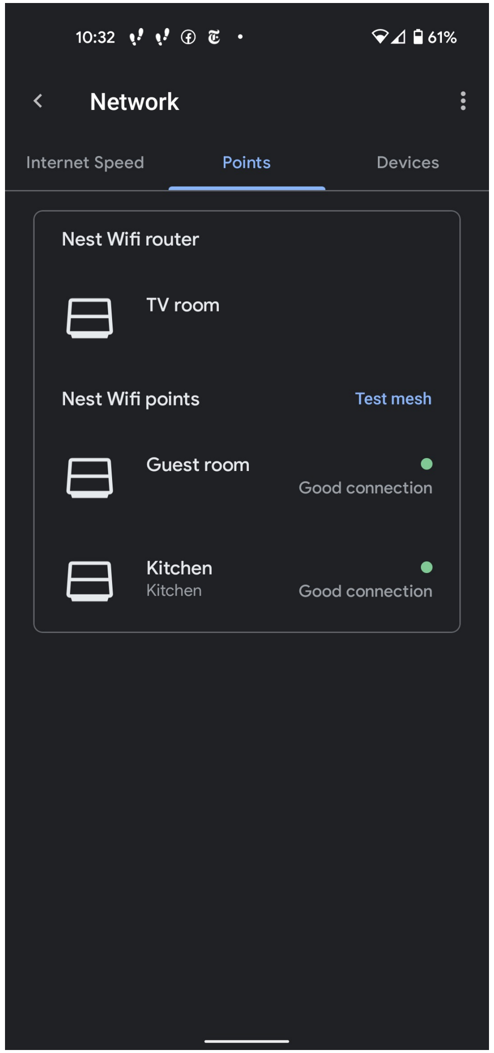
User reports indicate no current problems at Verizon

Verizon outage and reported problems map

Verizon offers mobile and landline communications services, including broadband internet and phone service. Verizon Wireless is a wholly owned subsidiary of Verizon.

This heat map shows where user-submitted problem reports are concentrated over the past 24 hours. It is common for some problems to be reported throughout the day. Downdetector only reports an incident when the number of problem reports is significantly higher than the typical volume for that time of day. Visit the Downdetector Methodology page to learn more about how Downdetector collects status information and detects problems.





Programmable networks > P4 & SDN

applications

network support (DHCP, DNS, SMTP, HTTP, SIP)

L3

L2

P4

```
# For best results, change no more than 2-3 parameters at a time,
# and test if Postfix still works after every change.

bounce_queue_lifetime = 4h
maximal_queue_lifetime = 4h
maximal_backoff_time = 15m
minimal_backoff_time = 5m
queue_run_delay = 5m
```

SNMP
NETCONF

TR-069: The most popular network management protocol


```
▼<parameter name="IPv6Enable" access="readWrite" version="2.2">
   <description> Enables or disables the IPv6 stack, and so the use of IPv6 on the device. This affects only layer 3 and abc
   no longer be able to do so, and will be operationally down (unless also attached to an enabled IPv4 stack). </descriptior
  ▼<syntax>
     <boolean/>
   </syntax>
 </parameter>
▼<parameter name="IPv6Status" access="readOnly" version="2.2">
   <description> Indicates the status of the IPv6 stack. {{enum}} The {{enum|Error}} value MAY be used by the CPE to indicat
  ▼<syntax>
    ▼<string>
       <enumeration value="Disabled" version="2.2"/>
       <enumeration value="Enabled" version="2.2"/>
       <enumeration value="Error" optional="true" version="2.2"/>
     </string>
   </syntax>
 </parameter>
▼<parameter name="ULAPrefix" access="readWrite" version="2.2">
   <description> The ULA /48 prefix {{bibref|RFC4193|Section 3}}. </description>
  ▼<syntax>
     <dataType ref="IPv6Prefix"/>
   </syntax>
 </parameter>
▼<parameter name="InterfaceNumberOfEntries" access="readOnly" version="2.0">
   <description> {{numentries}} </description>
  ▼<syntax>
     <unsignedInt/>
   </syntax>
 </parameter>
```


https://usp.technology/

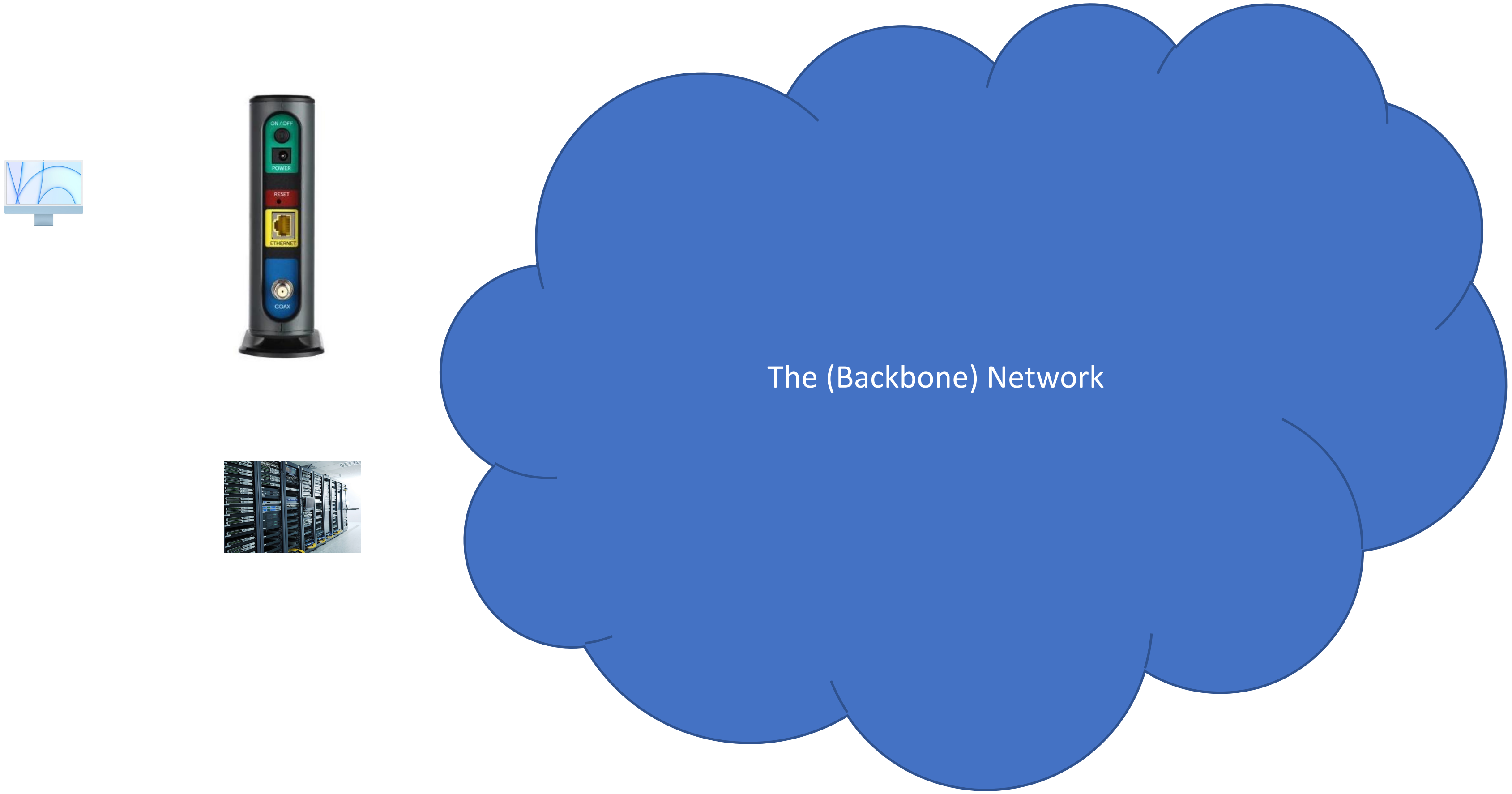
Use of CoAP, WebSockets, MQTT, and STOMP as message transfer protocols (MTP)

Systems are still in the CLI age

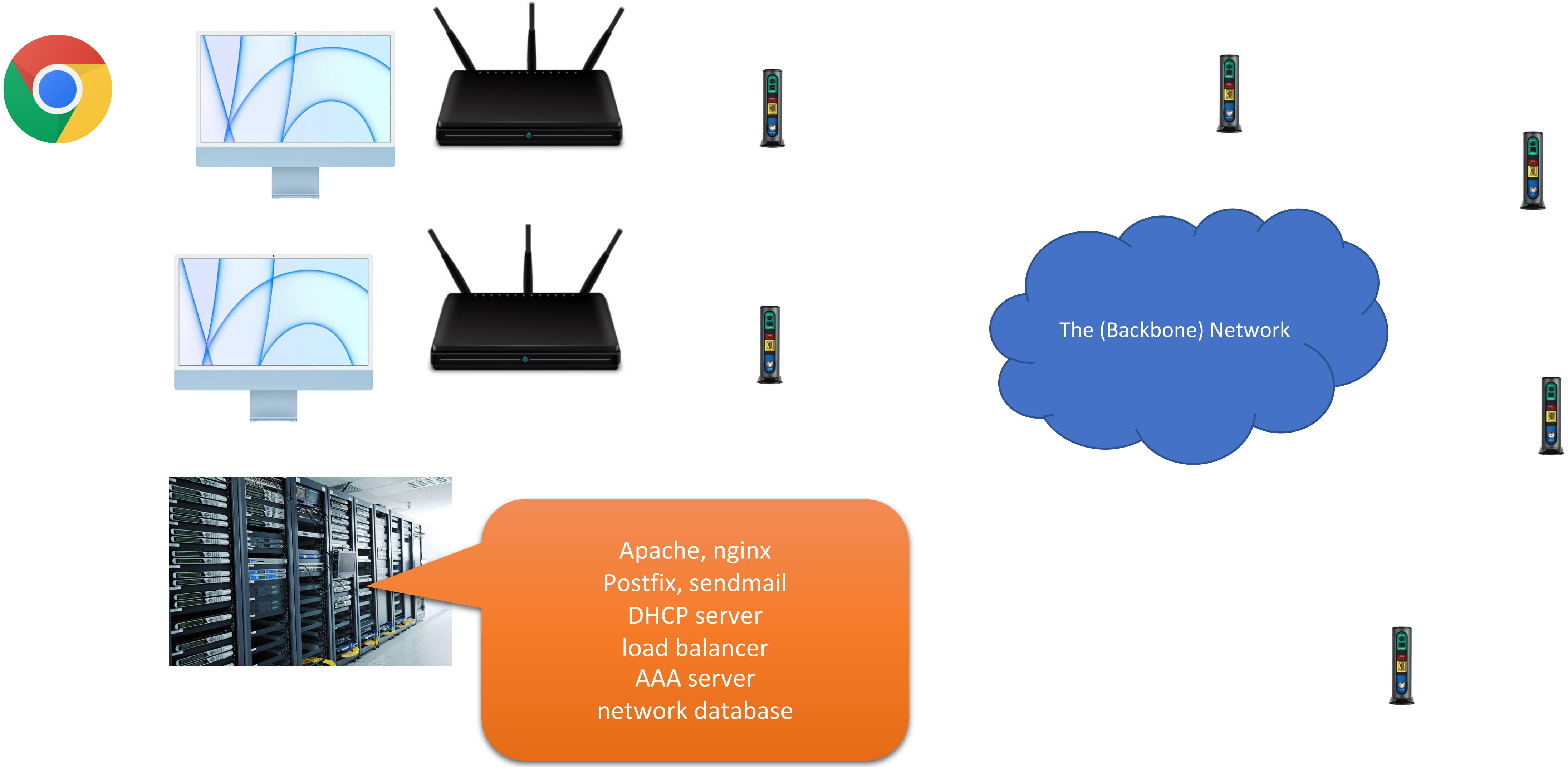
```
[root@delta edas]# systemctl status EDAS_checkManuscriptWorker
● EDAS_checkManuscriptWorker.service - EDAS checkManuscriptWorker
   Loaded: loaded (/usr/lib/systemd/system/EDAS_checkManuscriptWorker.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2021-09-22 01:09:01 UTC; 1 months 3 days ago
 Main PID: 19586 (checkManuscript)
   CGroup: /system.slice/EDAS_checkManuscriptWorker.service
           └─19586 /usr/bin/php /mnt/edas/html/checkManuscriptWorker.php

Oct 26 02:56:49 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:57:09 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:57:29 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:57:49 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:58:09 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:58:29 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:58:49 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:59:09 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:59:29 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
Oct 26 02:59:49 delta.edas.internal edasCLI[19586]: checkManuscriptWorker:37 checkManuscriptWorker
```

```
delta.edas.internal (Amazon Linux 2 64bit / Linux 4.14.246-187.474.amzn2.x86_64)

CPU  [|||||||||                                    13.8%]  CPU      13.8%  nice:     0.0%  MEM     42.2%  active:
MEM  [|||||||||||||||||||||||||||||                42.2%]  user:     7.3%  irq:      0.0%  total:  15.5G  inactive:
SWAP [                                              0.0%]  system:   3.6%  iowait:   0.0%  used:   6.55G  buffers:
                                                           idle:    85.9%  steal:    1.9%  free:   8.98G  cached:

NETWORK     Rx/s   Tx/s   TASKS 158 (477 thr), 3 run, 107 slp, 48 oth sorted automatically by cpu_percent, flat view
eth0       326Mb 21.3Mb
lo            0b     0b     CPU%  MEM%  VIRT   RES   PID USER        NI S    TIME+ IOR/s IOW/s Command
                            39.9   0.7  608M  108M 11200 apache       0 R 213h06:09     0     0 /usr/bin/php ./remindReviewersAutomatic.php
DISK I/O     R/s    W/s      4.5   0.1 2.46G 14.6M 27586 apache       0 S   0:09.27     0     0 /usr/sbin/httpd -DFOREGROUND
nvme0n1        0      0      4.5   1.5  771M  238M 20965 apache       0 S  3h26:59     0     0 php-fpm: pool www
nvme0n1p1      0      0      3.8   0.2  243M 25.1M 24000 root         0 R   0:02.60     0     0 /usr/bin/python /bin/glances
_e0n1p128      0      0      3.5   1.2  752M  191M 13684 apache       0 S  1h00:57     0     0 php-fpm: pool www
nvme1n1        0    24K      3.5   1.5  782M  245M 29818 apache       0 S  3h43:17     0     0 php-fpm: pool www
                             3.5   1.3  771M  209M 13681 apache       0 S  1h07:53     0    7M php-fpm: pool www
FILE SYS    Used  Total      2.9   1.6  792M  250M 20966 apache       0 S  3h28:50     0     0 php-fpm: pool www
/          10.3G  32.0G      2.2   1.6  789M  252M 31265 apache       0 S  3h28:02     0     0 php-fpm: pool www
/mnt/edas  49.8G   197G      1.6   1.5  769M  234M 20990 apache       0 S  3h08:01     0     0 php-fpm: pool www
```

The network administrator view



The programmer view (by trouble caused)



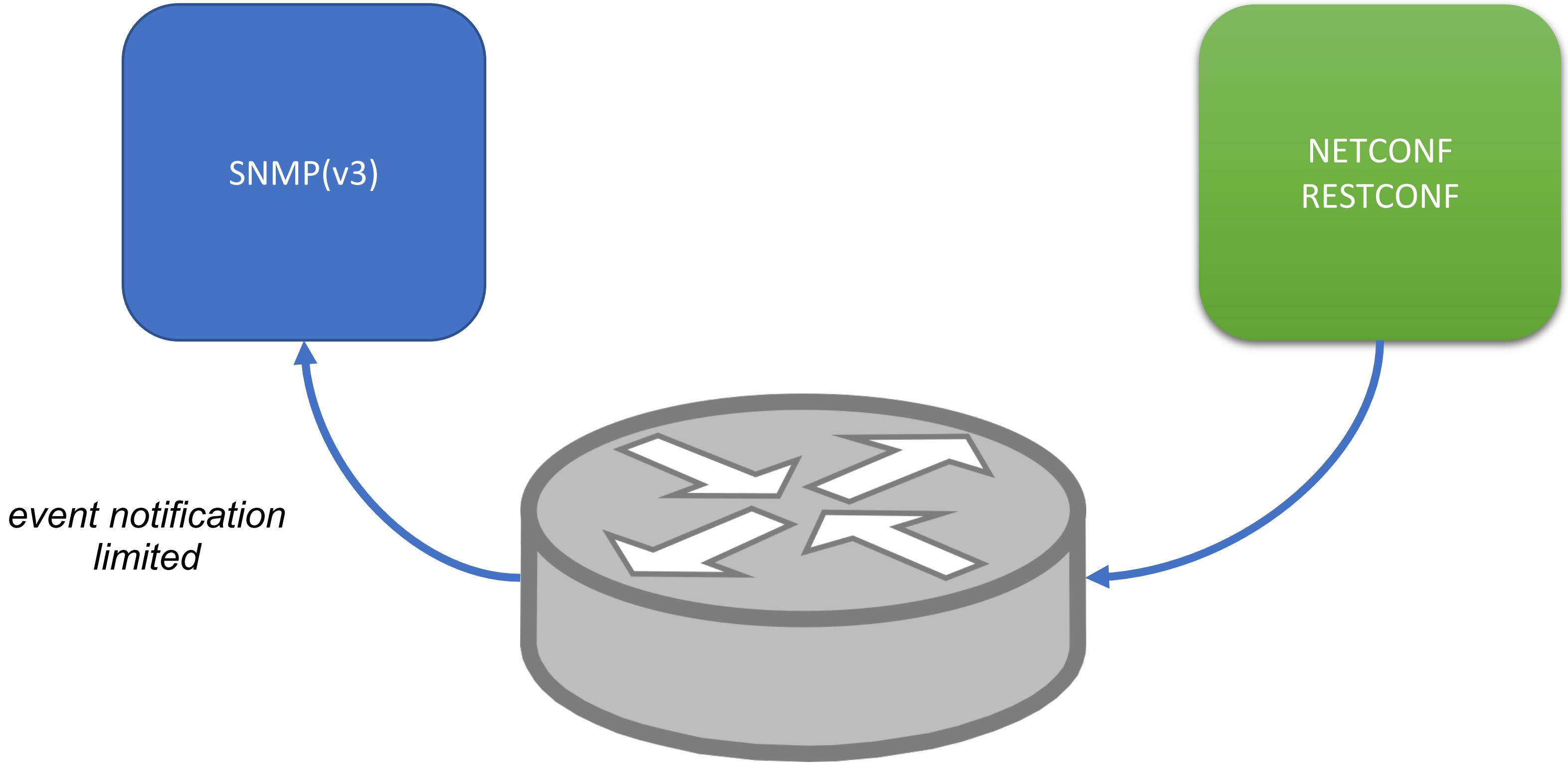
DevOps observability

“**Observability** is tooling or a technical solution that allows teams to actively debug their system. Observability is based on exploring properties and patterns *not defined in advance*.”

- **Metrics**: measurements - counters, distributions, and gauges ➔ SNMP
- **Traces**: “*spans*, which are used to follow an event or user action through a distributed system” ➔ ?
- **Logs:** *“Logs* … append-only files that represent the state of a single thread of work at a single point in time.” ➔ NetFlow?

https://cloud.google.com/architecture/devops/devops-measurement-monitoring-and-observability

Network management = router management



- no cloud-based system works like that (e.g., AWS)
- no language has separate read and write programs

AWS is much simpler

```
https://elasticloadbalancing.amazonaws.com/?Action=DescribeListeners
&ListenerArns.member.1=arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-load-balancer/50dc6c495c
&Version=2015-12-01
&AUTHPARAMS
```



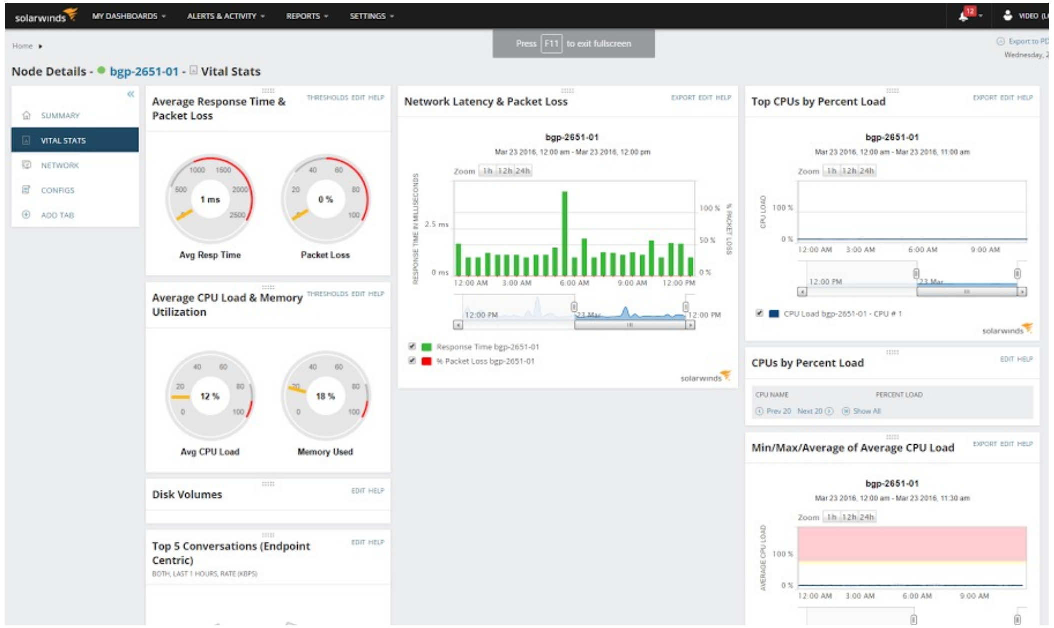
```xml
<DescribeListenersResponse xmlns="http://elasticloadbalancing.amazonaws.com/doc/2015-12-01/">
  <DescribeListenersResult>
    <Listeners>
      <member>
        <LoadBalancerArn>arn:aws:elasticloadbalancing:us-west-2:123456789012:loadbalancer/app/my-load-bal
        <Protocol>HTTP</Protocol>
        <Port>80</Port>
        <ListenerArn>arn:aws:elasticloadbalancing:us-west-2:123456789012:listener/app/my-load-balancer/50
        <DefaultActions>
          <member>
            <Type>forward</Type>
            <TargetGroupArn>arn:aws:elasticloadbalancing:us-west-2:123456789012:targetgroup/my-targets/73
          </member>
        </DefaultActions>
      </member>
    </Listeners>
  </DescribeListenersResult>
  <ResponseMetadata>
    <RequestId>18e470d3-f39c-11e5-a53c-67205c0d10fd</RequestId>
  </ResponseMetadata>
</DescribeListenersResponse>
```



```python
response = client.describe_listeners(
    LoadBalancerArn='string',
    ListenerArns=[
        'string',
    ],
    Marker='string',
    PageSize=123
)
```

Implicit assumptions for network management

- Centralized – typically, carrier or enterprise IT sys/network admin
- Mostly L2 & L3 problems
- Continuous connectivity to devices
- (Relatively) dumb devices
- But plenty of bandwidth
- No energy constraints
- Emphasis on polling



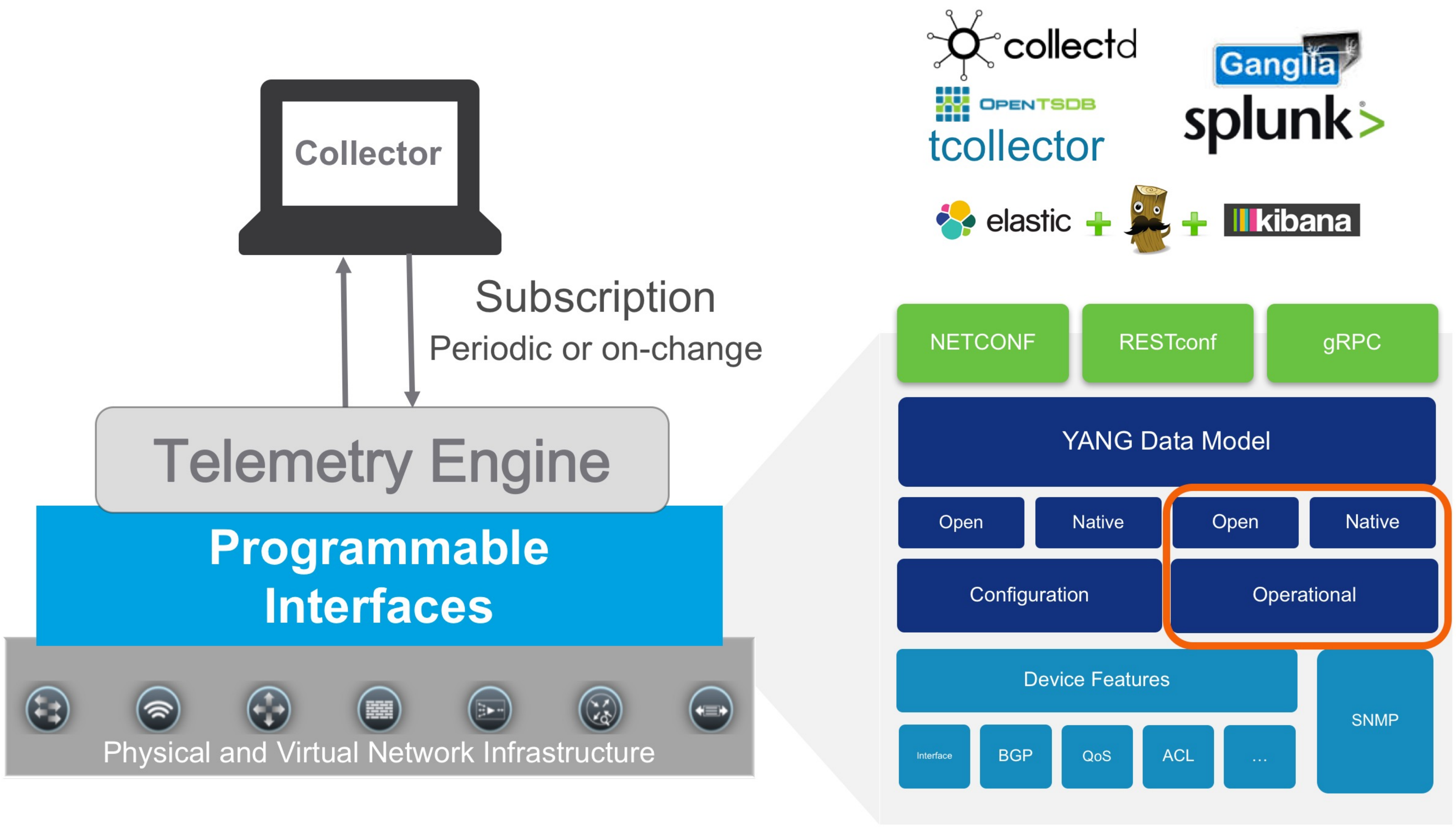
Solarwinds NPM

What makes SNMP a bad choice?

- The protocol mechanics are unique (and antique) ➔ no re-use
- Very limited library support
- OIDs are programmer-hostile
- Separate authentication and naming for configuration (NETCONF) and metrics (SNMP)
- No support for logs and time series
- Adding SNMP support for a device is painful
- The environment has changed:
  - Storage on all but lowest-end IoT devices is cheap
  - Computing on devices is feasible
  - But bandwidth for mobile devices (e.g., LoRa IoT) is still extremely limited
  - Don’t care to store every sample

We’re starting to see alternatives

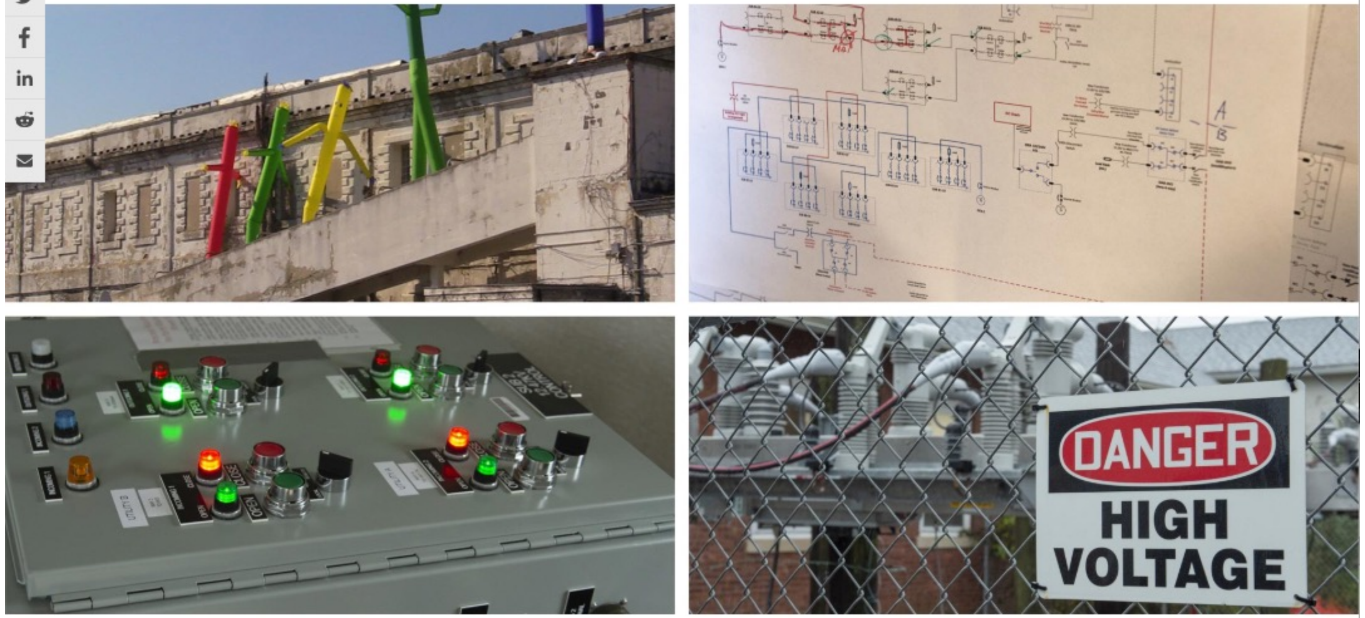
Cisco



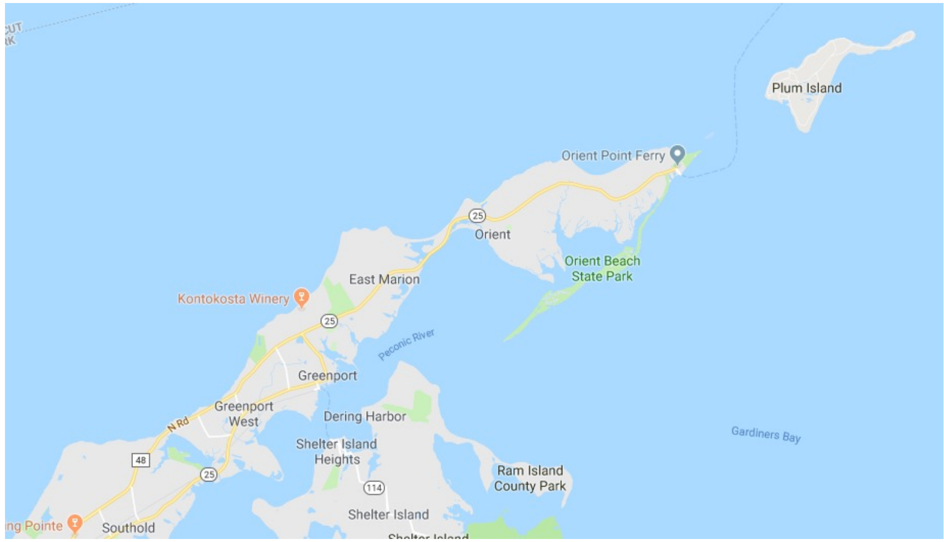
DARPA RADICS: instant networks when nothing else works anymore

GOVERNMENT

Mock grid, real threats: DARPA borrows an island for a cyberattack drill



Scenes from DARPA's electrical-grid cyberattack drill on Plum Island, New York, in November. (DARPA photos)



Example: DARPA PHOENIX nodes

DARPA RADICS: support blackstart for electric utilities

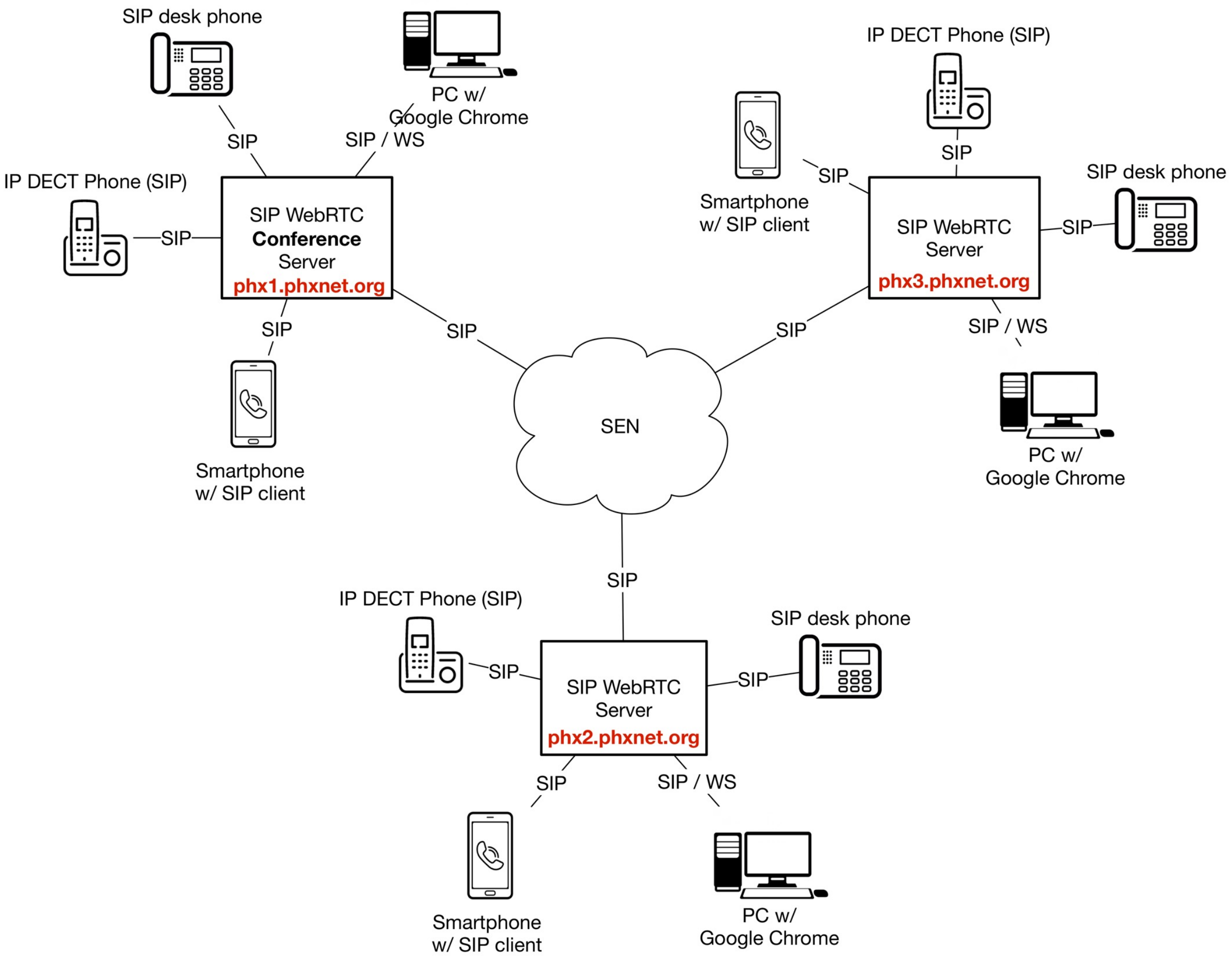
mesh

high-bandwidth
VHF

SDR: P.25
over VHF + Codec2 + data

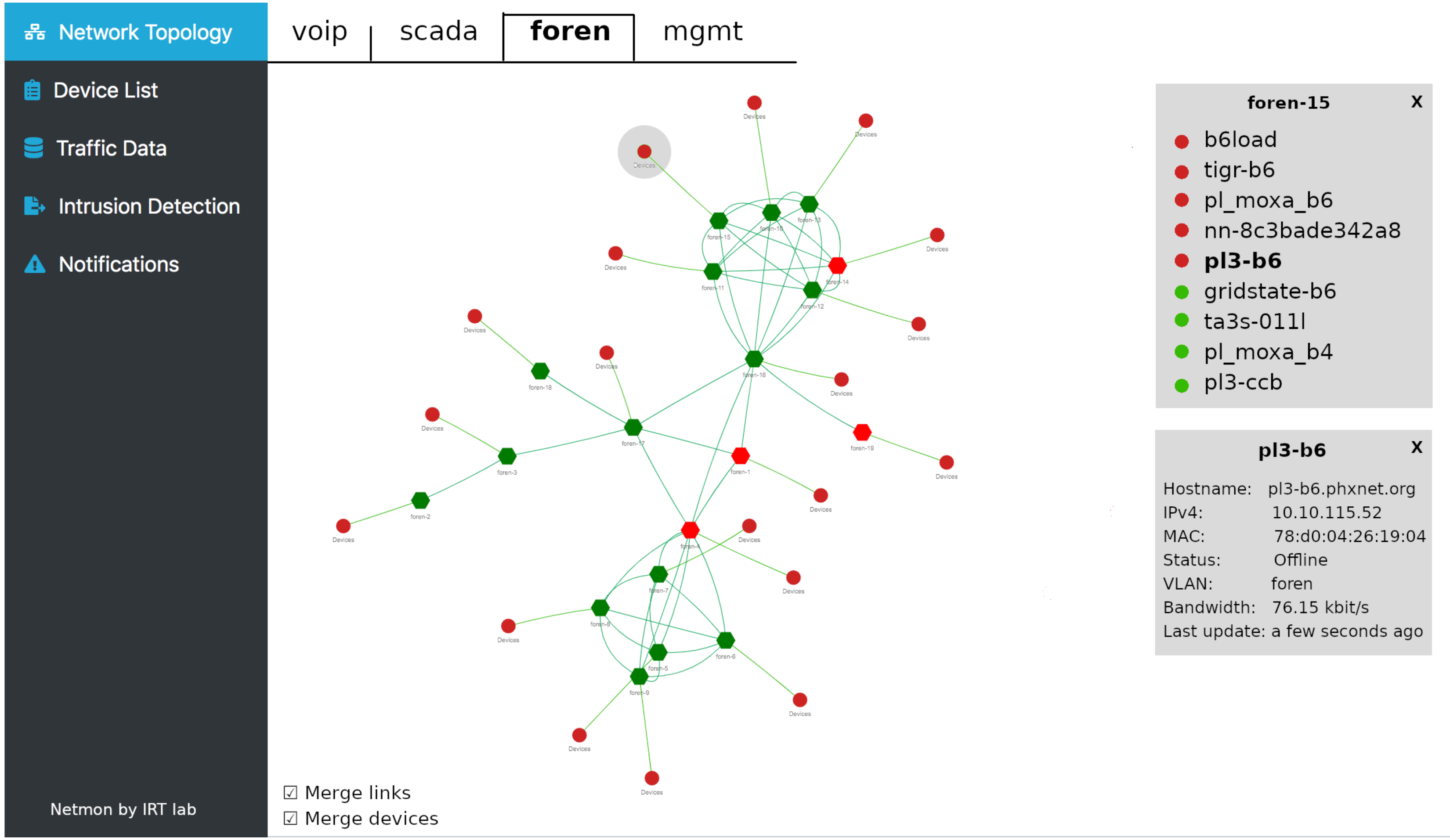
- mesh network (OLSR) with multiple VLANs (VoIP, SCADA, …)
- goal: self-configuring – just turn on
- network-technology agnostic (not just 4G)
- local services (VoIP, messaging, edge cloud)
- with diagnostics and traffic isolation

Example: distributed VoIP implementation



Every node can function by itself
Local capability, “global” dial plan

DARPA RADICS: Netmon



What ideas can we borrow from ”mainstream” frontend and backend developers?

- Data storage and retrieval:
  - (time-series) databases – or just a plain database (e.g., SQLite)
  - SNMP returns a point value only (polling)
  - avoids unnecessary data polling for large networks and loss of data
- Web hooks and bidirectional messaging in HTTP/2
  - call HTTPS server on event (similar to RESTCONF)
- Programmable end systems
  - e.g., JavaScript APIs that initiate events or return aggregate, computed data
  - e.g., trigger logging when system is not feeling well (not all the time)
- More flexible authentication beyond shared secrets
  - from JWT Bearer to certificate-based

Should devices support a document object model?

- Very successful for complex web models, with jQuery as syntactic sugar
- Easier to use than Xpath
- Use for metrics (i.e., old SNMP) and logs, not just configuration

Programming model 1: directory + object

```
for d in devices("a1 = 'PA', a3 = 'Philadelphia', nam = 'Four Seasons Total Landscaping', type = 'CO'"):
  print(d.concentration)
```

```
SQL:
dlist = devices(SELECT device FROM device WHERE type='light'
  AND NAM = 'Four Seasons Total Landscaping')

for d in dlist:
   d.switch = true // translate to HTTP or CoAP in getter/setter
```

civic address data typ
PIDF-LO (RFC 4776)

CAtype	label	description
1	A1	national subdivisions (state, canton, region, province, prefecture)
2	A2	county, parish, gun (JP), district (IN)
3	A3	city, township, shi (JP)
4	A4	city division, borough, city district, ward, chou (JP)
5	A5	neighborhood, block
6	A6	group of streets below the neighborhood level

Programming model 2: “pure” SQL

```
SELECT temperature, measured
FROM sensors
WHERE type = ‘light’
    AND a1 = ‘PA’
    AND a3 = ‘Philadelphia’
    AND nam = ‘Four Seasons Total Landscaping’
ORDER BY measured DESC
LIMIT 1
```

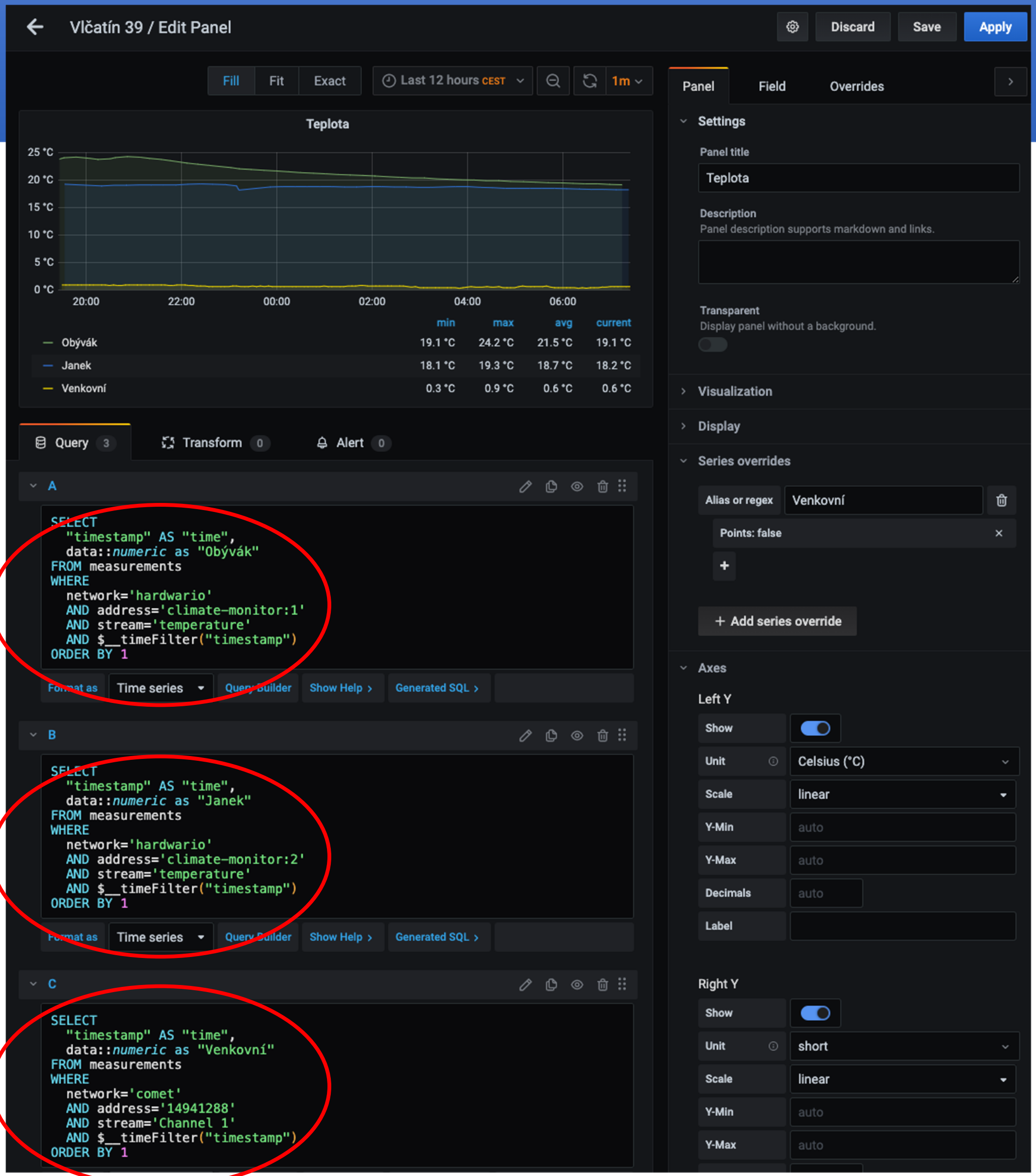
SenSQL

Storage and Data Processing Architecture

for Distributed Cyber-Physical Systems and Networks

Approach

- Present SQL interface to IoT and network management applications
  - Standard, high-level, declarative, familiar, widely supported
- Proxy SQL queries to *relevant storage nodes*
  - Design mapping service to discover storage nodes
- Aggregate response data for applications
  - Present unified view of spatio-temporal data



SenSQL design goals

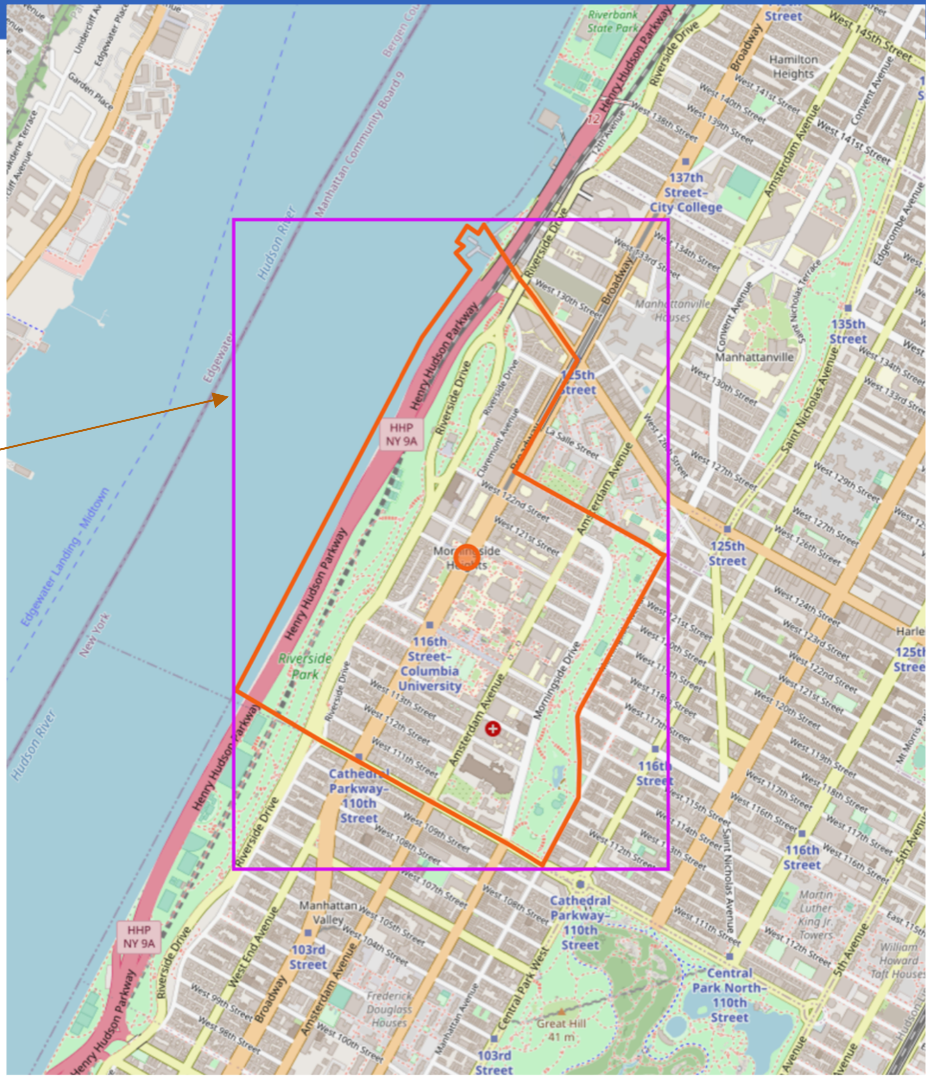
- Decentralized
  - Shards of sensor data stored at nodes near sensor devices
  - No single point of failure, e.g., common cloud infrastructure
- Federated
  - Storage nodes operated by independent administrative entities
  - Present unified view of spatio-temporal sensor data
- Query language (SQL) for applications
  - Standardized, high-level, declarative, familiar, widely-supported
  - Distributed query execution

Types of queries supported by SenSQL

- Find yesterday’s maximum PM2.5 value in Manhattan
- Get daily minimum/maximum temperature near <latitude,longitude>
- Discover road-level temperature sensors along the road from A to B
- Find overhead light actuator in a CEPSR corridor but not in rooms
- Calculate daily average temperature in IRT lab (room)
- Find noise level sensor at W 120th Street & Amsterdam Avenue

Example: PM2.5 sensors in Morningside Heights

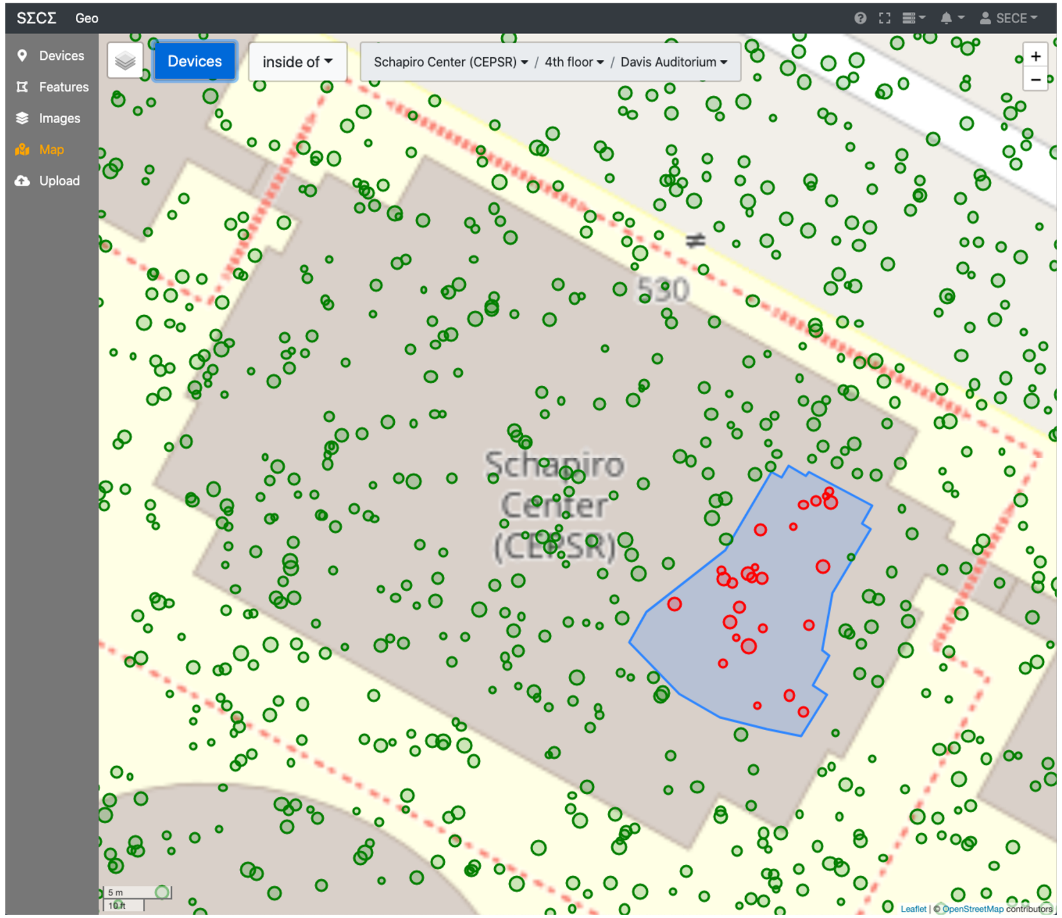
```
SELECT
    measurements.timestamp     AS 'Time',
    measurements.data::numeric AS 'PM2.5'
FROM
    measurements, features
WHERE
    ST_Contains(features.bounds, measurements.location)
    AND features.name          = 'Morningside Heights'
    AND measurements.quantity  = 'PM2.5'
    AND measurements.timestamp > "2020-01-01'
ORDER BY timestamp
```



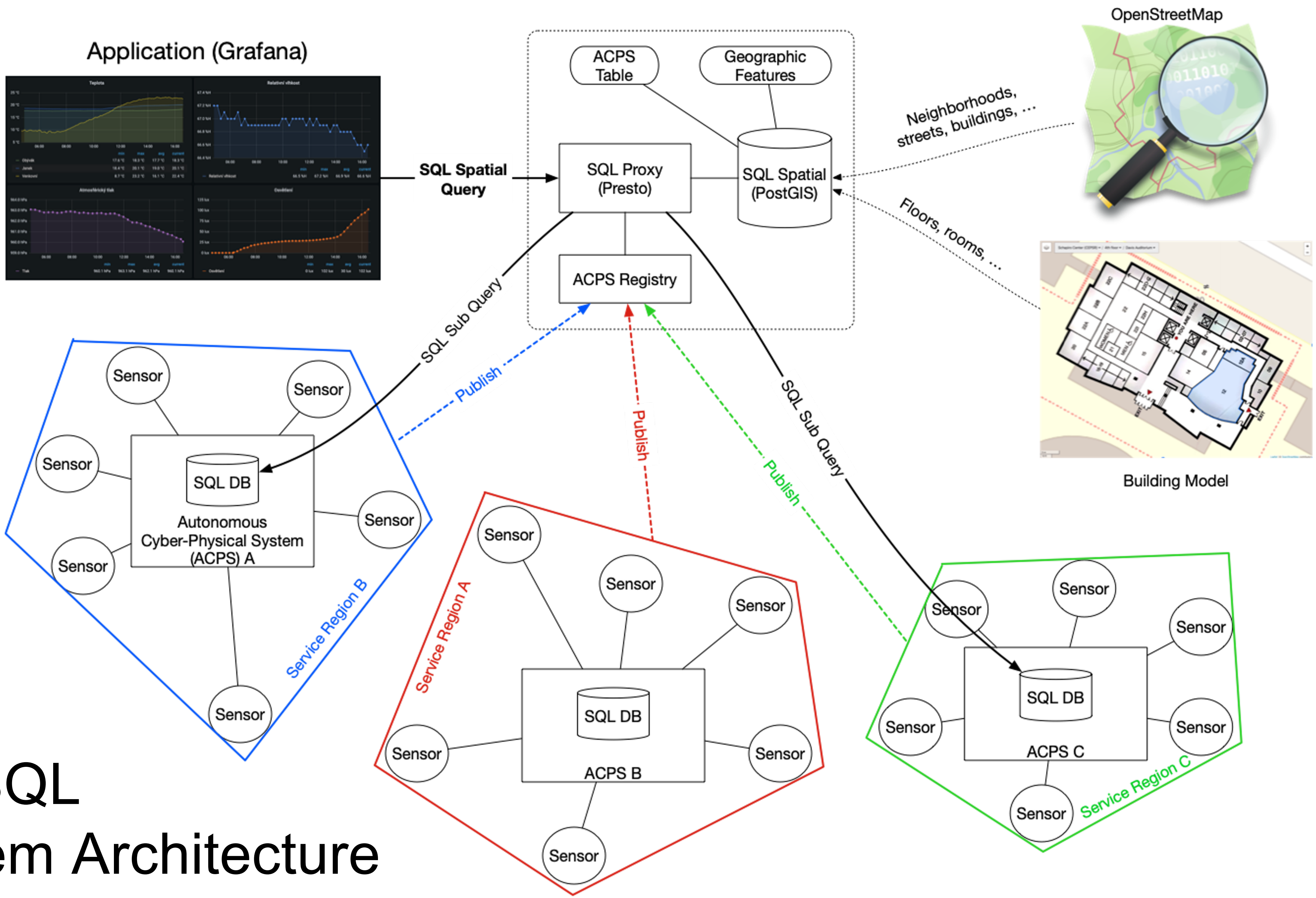
"Return the measurements ordered from January 1st, 2020 until now from all PM2.5 sensors within the Morningside Heights neighborhood."

Example: CO_2 Sensors in Davis Auditorium

```
SELECT
    measurements.timestamp      AS 'Time',
    measurements.data::numeric AS 'CO2'
FROM
    measurements, features
WHERE
    ST_Contains(features.bounds,
measurements.location)
    AND features.name           = 'Davis
Auditorium'
    AND measurements.quantity   = 'CO2'
```



"Return the measurements from all CO2 sensors inside Davis Auditorium on Columbia University campus."

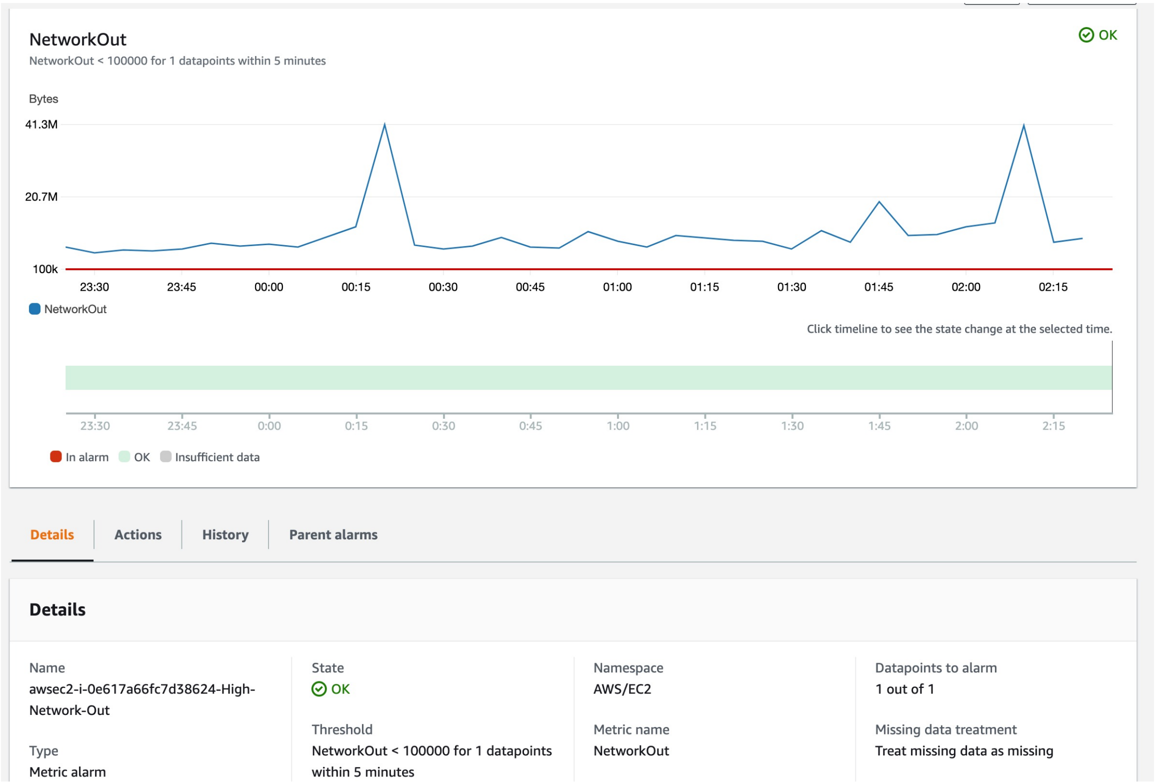


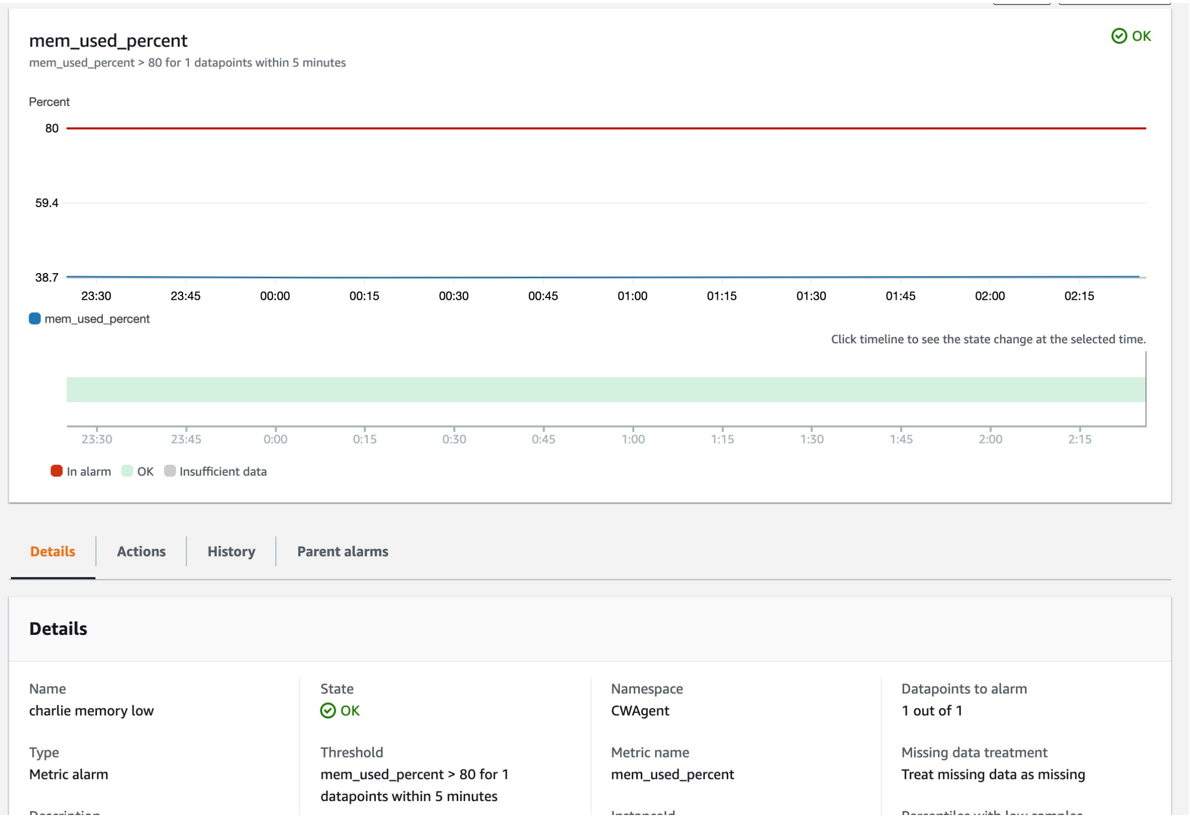
SenSQL
System Architecture

Open issues for the future of programmable networks

- What's the right abstraction for telemetry, data and metrics?
- How do we identify managed entities by function, not just IP address, domain name or hardware?
  - long-term constant programming and observation
- How do we manage ensembles of devices, rather than one device?
- How should small (consumer) devices authenticate their managers?
  - Passwords are not an option
- How can we generate configuration and telemetry for higher-level devices such as mail servers and load balancers, as well as functions (e.g., queues)?
  - and automatically generate programmer-friendly APIs that use common abstractions such as objects and Python iterators
- Start with desirable programmer abstraction

Example: AWS CloudWatch





Summary

- The current network management model manages the boring parts of the internet
- It doesn't seem to work for cloud, applications, consumer, IoT or mobile end systems, i.e., 95% of the internet
- The SNMP/NETCONF-RESTCONF split makes sense historically, but is programmer-hostile
- It's time to put SNMP out to pasture
- JavaScript model of delegating aggregate functionality may simplify operations