How to pythonize the data from Cisco WLC

Roman Podoynitsyn

The goal

- To introduce the Python parser for WLC config (aka wlc-pythonizer)
- To discuss the use cases available and gather ideas for future
- To demonstrate how it works
- To invite for collaboration

How it all started

The relationships between engineer and WLC config

- We often work with WLC config files
- These files contain a lot of semi-structured data in text format
- The information is really useful but hard to grasp in text editor
- Love it or hate it?

Project goals

- Solve the problems asked by my Customers
- Save all historical configuration and operational data in easy-to-use format (export as pickle, json, yaml etc.)
- Get quick answer on my fingertips (easy to use filters)
- Be able to visualize data to find hidden patterns
- Improve Python skills and have fun!!!
- Try AI capabilities on collected dataset (*future releases)

Some disclaimers

- Work in progress, not a complete product or service to use
- Many undiscovered bugs may exist
- Some scalability tests were done (45 Mb config with ~2500 AP)
- Not a python guru, sometimes the code is awkward
- Tested with Python 3.7
- No fancy GUI, but some nice visualizations

Why use customized objects?

- Clear representation
- Context help to quickly find attribute
- Human-readable syntax
- Customized methods (like compare whole config or its special part, filter by attribute or value)
- Fast customization (if needed)

Winning combination for any config analysis task

- Your wireless network skills
- Basic knowledge of Python

Examples of use cases

Some examples that are easy to solve with tool

- What exactly changed in the config since the last (day, week, month)?
- Best practice rules – update, adjust and customize for your Cu
- Do we have the same SSID settings in every branch?
- How diverse is network configuration for 100 WLCs installed in Customer network?
- Which rogue APs have the most impact on our network?
- Etc.

And eventually…
It’s time for DEMO

Demo scenarios

- Parse config file (if you do not have your own config files, you can use config samples as suggested in tutorial)
- Explore data with context help (refer to object tree)
- Define best practice rules and check WLC config against them
- Compare SSID config for different WLCs
- Compare SSID configs of the same WLC collected in different time
- Config diversity
- Explore rogue APs
- Explore channel utilization and its impact factors

Parse config

The ways to import config

- From file

```
wlcs = parse_file('wlc_config_example.txt’)

Following WLCs were parsed from file:  ['tac-test’, ‘wlc2’]
```

The ways to import config – via SSH

- Via SSH

```
wlcs = ssh_collect()
Collecting config from WLC via SSH
WLC IP address, please: >? 10.11.12.13
Your username, please: >? admin
Your password, please: >? *******
sending username...
sending pwd...
Please, hold on, getting config, it can take some time...
Collection is completed
Please, hold on, parsing configs, it can take some time...
Configs are written to file with name:  WLC_config-IP 10.11.12.13-
Tue Aug 7 111500 2021.txt
Configs are sent to parser….
Following WLCs were parsed…
```

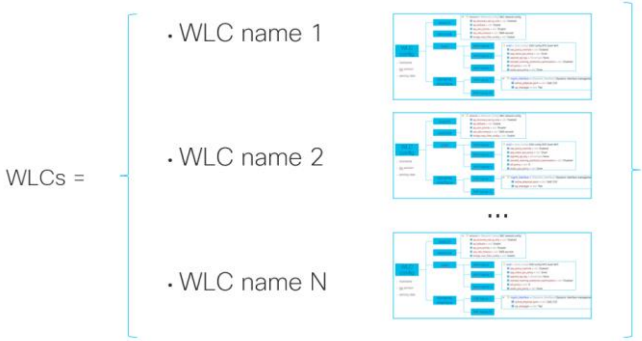
The ways to import config – via Cisco DNA Center

- Via DNAC API

```
wlcs = dnac_get_wlc_configs()
Enter the IP address of DNAC, please: >? 192.168.44.44
Your username, please: >? user
Your password, please: >? ****
Contacting Cisco DNA Center with IP address:  192.168.44.44
for username:  user
Following WLC devices are found and reachable:
sdc1wlc001
Starting config collection...
Waiting for collection results ...
DNA Center collected configs, grabbing config files from it...
WLC config files are successfully collected
Configs are written to file with name:  all_WLC_config-DNAC-
Configs are sent to parser
...
Following WLCs were parsed from file:  ['sdc1wlc001']
```

To access the WLC config – refer by hostname

- Files may contain the config from many WLCs
- All the functions above return the Python dictionary, keys are hostnames



WLCs =

- WLC name 1
- WLC name 2

...

- WLC name N

```
To access the WLC config objects:
wlc1 = wlcs[<wlc1 hostname>]
wlc2 = wlcs[<wlc2 hostname>]
```

Explore the data

The ways to explore parsed data

- Just print the object name in console

```
wlc

WLC config for host: WC-Cisco-CX-020, version is 8.10.142.0,
collection time is Sun Jul  7 00:00:07 2021,
parsing date is Mon Aug 12 152115 2021
```

The ways to explore parsed data

- Add attributes after point (config section type 1)

```
wlc.network

WLC network config

wlc.switch

WLC switch config
```

The ways to explore parsed data

- Add attributes after point (config section type 2)

```
wlc.dynamic_interfaces

Named List of length 7 with items Dynamic_Interface: management,
redundancy-management,  redundancy-port,  service-port,  virtual,
wifi_guest,  wifi_guest_bras,  wifi_guest_old, wifi_cisco,
wifi_users,  wifi_users_vip, wifi_voice

wlc.ssid

Named List of length 7 with items Ssid_Config: Cisco_WLC,
9800_best,  AireOS, BEST-WIFI, Provision, Corporate, Guest WiFi
```

The ways to explore parsed data

- Add [name] to get the item from list (config section type 2)

```
wlc.dynamic_interfaces['management’]

Dynamic interface management

wlc.ssid[‘Guest’]

SSID config Guest
```

The ways to explore parsed data

- Add point and the attribute name to get the value

```
wlc.dynamic_interfaces['management'].ip_address

'192.168.176.129’

wlc.dynamic_interfaces['management'].active_physical_port
'LAG (13)'
```

Hint: Don’t remember the name of attribute? Stay calm and start printing!

(Tab button is your friend for context help)

List is too long? - Use “filter” method

```
SYNTAX:
Object.filter(<ATTRIBUTE NAME>,<VALUE>)

wlc.ap_rf

Named List of length 651 with items Ap_Rf_Config: WA-KUR-KRCH-001-
1100_slot0, …

#Find all radios with failed noise profile
wlc.ap_rf.filter('noise_profile','FAILED’)

Named List of length 2 with items Ap_Rf_Config:
WA-MSK-CWK_slot0,  WA-MSK-SBC-002_slot0

#Double filtering is allowed too ;)
wlc.rogue_aps.filter('channel','11').filter('state','Alert')
```

Display all values? – Use “show” command

```
SYNTAX:
show(object)
object.show()

#Display all attributes of network config section
wlc.network.show()
...
rf_network_name wifimsk
dns_server_ip 192.168.0.41
web_mode Disable
secure_web_mode Enable
secure_web_mode_cipher_option_high Enable
secure_web_mode_ssl_protocol Disable
web_csrf_check Enable
ocsp Disabled
...
```

Find attribute or value? – Use “grep” command

```
SYNTAX:
grep(object,’value’)
object.grep(‘value’)

#Find all IPv6 MLD parameters
wlc.grep('mld’)
WLC network config mld_snooping Disabled
WLC network config mld_timeout 60 seconds
WLC network config mld_query_interval 20 seconds
#Find all timeouts for SSID
wlc.ssid[‘Guest'].grep('second’)
SSID config Guest exclusionlist_timeout 60 seconds
SSID config Guest session_timeout 1800 seconds
SSID config Guest scan_defer_time 100 milliseconds
#Find all radios with FAILED profiles
wlc.ap_rf.grep('FAIL’)
AP RF config WA-MSK-SSH-001_slot0 interference_profile FAILED
AP RF config WA-MSK-TEL-003_slot0 noise_profile FAILED
```

Using list comprehension - 1

```
[x for x in range(6)]

[0, 1, 2, 3, 4, 5]
```

```
[ssid.name for ssid in wlc.ssid] # get all names of SSIDs configured

['Guest_WiFi’,
 'Users',
 ‘9800-1x',
 ‘9800-EOGRE',
 'Provision',
 'Corporate’,
 ‘Cisco Guest WiFi']
```

Using list comprehension - 2

```
[(x,x*2) for x in range(6)] # get pairs of number, doubled number

[(0, 0), (1, 2), (2, 4), (3, 6), (4, 8), (5, 10)]
```

```
# get name and status of SSIDs configured
[(ssid.name, ssid.status) for ssid in wlc.ssid]

[('Guest_WiFi', 'Disabled'),
 (‘9800-1X', 'Enabled'),
 ('Users', 'Enabled'),
 ('Provision', 'Enabled'),
 ('Corporate', 'Enabled'),
 ('Guest WiFi', 'Enabled')]
```

Using list comprehension - 3

```
[(x,x*2) for x in range(10) if x%2 ==0] # get pairs for odd numbers

[(0, 0), (2, 4), (4, 8)]
```

```
# get the names of SSIDs in Disabled status
[ssid.name for ssid in wlc.ssid if ssid.status == 'Disabled’]

['Guest_WiFi’]
# get the names of SSIDs with AAA override
[ssid.name for ssid in wlc.ssid
    if ssid.aaa_policy_override == 'Enabled’]

# get the names of SSIDs with CWA
[ssid.name for ssid in wlc.ssid
   if ssid.aaa_policy_override == 'Enabled’ and
   'Guest' in ssid.name]
```

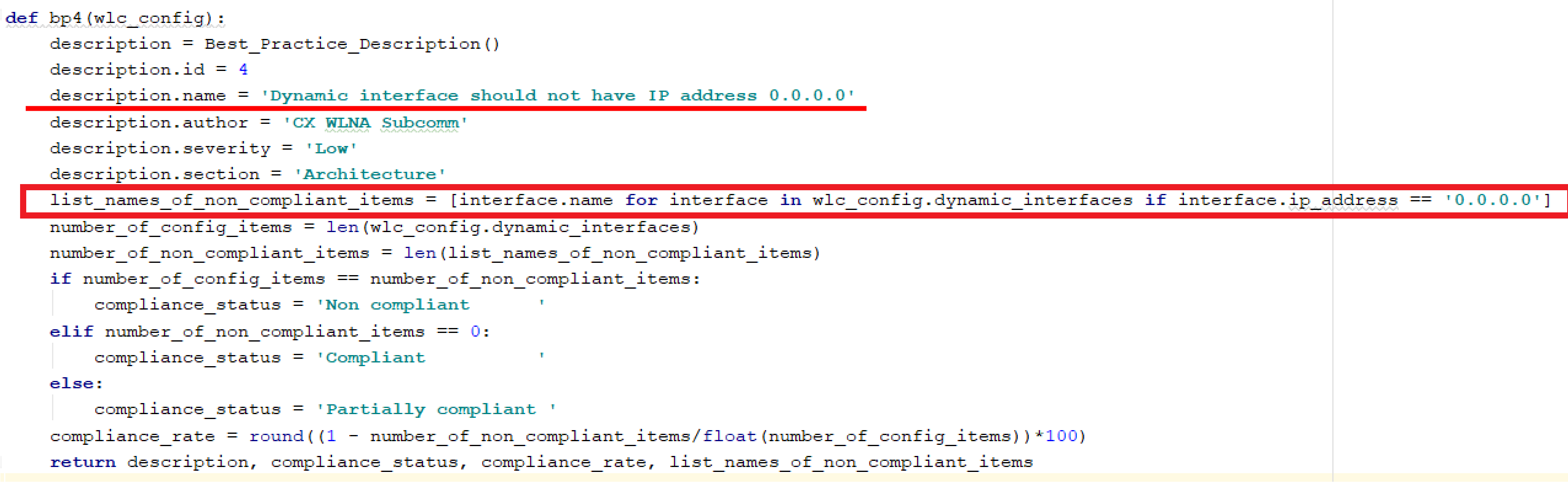
Best practice check

To check the BP rule – use tools available

- Translate the BP rule into Python code
- Usually one string is enough
- Example: ‘Telnet service should be disabled on AP’

```
#Get all AP names with telnet enabled

[ap.name for ap in wlc.ap_configs if 'Enable’ in ap.telnet_state]
```

Example of function to check BP rule



```
def bp4(wlc_config):
    description = Best_Practice_Description()
    description.id = 4
    description.name = 'Dynamic interface should not have IP address 0.0.0.0'
    description.author = 'CX WLNA Subcomm'
    description.severity = 'Low'
    description.section = 'Architecture'
    list_names_of_non_compliant_items = [interface.name for interface in wlc_config.dynamic_interfaces if interface.ip_address == '0.0.0.0']
    number_of_config_items = len(wlc_config.dynamic_interfaces)
    number_of_non_compliant_items = len(list_names_of_non_compliant_items)
    if number_of_config_items == number_of_non_compliant_items:
        compliance_status = 'Non compliant      '
    elif number_of_non_compliant_items == 0:
        compliance_status = 'Compliant          '
    else:
        compliance_status = 'Partially compliant '
    compliance_rate = round((1 - number_of_non_compliant_items/float(number_of_config_items))*100)
    return description, compliance_status, compliance_rate, list_names_of_non_compliant_items
```

To apply the set of BP rules – use “bp_check”

```
SYNTAX:
bp_check(wlc_config)

#Check WLC config against the set of rules
bp_check(wlc)
Best practices compliance report
ID Status               Rate Name
1 Compliant             100 Telnet should be disabled in all APs
2 Compliant             100 Each SSID is mapped with unique interface in controller if no AAA override is enabled
3 Compliant             100 Local Client profiling using HTTP and DHCP is enabled unless RADIUS profiling is in use
4 Compliant             100 Dynamic interface should not have IP address 0.0.0.0
5 Compliant             100 Primary and secondary DHCP server IP addresses are configured for WLC dynamic interfaces
```

Some advantages

- Relatively easy translation of business requirements into code
- Customize and implement if needed
- Fast automated checking
- Double-check values in the same tool if in doubt

Compare config files or its sections

Compare config? – Use “compare” function

```
SYNTAX:
compare(object1, object2)


#Get the same SSID config from the same device during last 3 weeks
ssid1 = archive['Week 31']['WC-MSK-CX'].ssid['Corporate']
ssid2 = archive['Week 32']['WC-MSK-CX'].ssid['Corporate']
ssid3 = archive['Week 33']['WC-MSK-CX'].ssid['Corporate’]

#Call compare function
a = compare(ssid1,ssid2)
Compare called for  SSID config Corporate
SSID config:
  Subdiffs:
  - 0   #Zero changes between Week 31 and 32
  - 196 #196 parameters were compared
```

Compare config? – Use “compare” function

```
SYNTAX:
compare(object1, object2)

#Call compare function
a = compare(ssid2,ssid3)
Compare called for  SSID config Corporate
SSID config:
  Subdiffs:
  - 3    #3 parameters were changed between Week 32 and 33
  - 196  #196 parameters were compared
  d_802_11k_neighbor_list: # <-The name of parameter
  - Disabled               # <-The value for Week 32
  - Enabled                # <-The value for Week 32
  d_802_11v_bss_transition_service:
  - Disabled
  - Enabled
  security_ft_support:
  - Disabled
  - Adaptive
```

“Compare” function

- Works with any config section
- Recursively calls every branch in config tree
- If called for list, compares only items with the same name, for example SSID ‘Corporate’ with SSID “Corporate’
- Calculates the number of parameters compared
- Calculates the number of differences in values

Know the diversity of configs

Compare without details – get the “big picture”

- Diversity metric defined as:

$$Diversity = \frac{Number\ of\ differences\ found}{Number\ of\ parameters\ analyzed}$$

- Can take values from:

0% - no differences, configs are completely identical

100% - all values are different

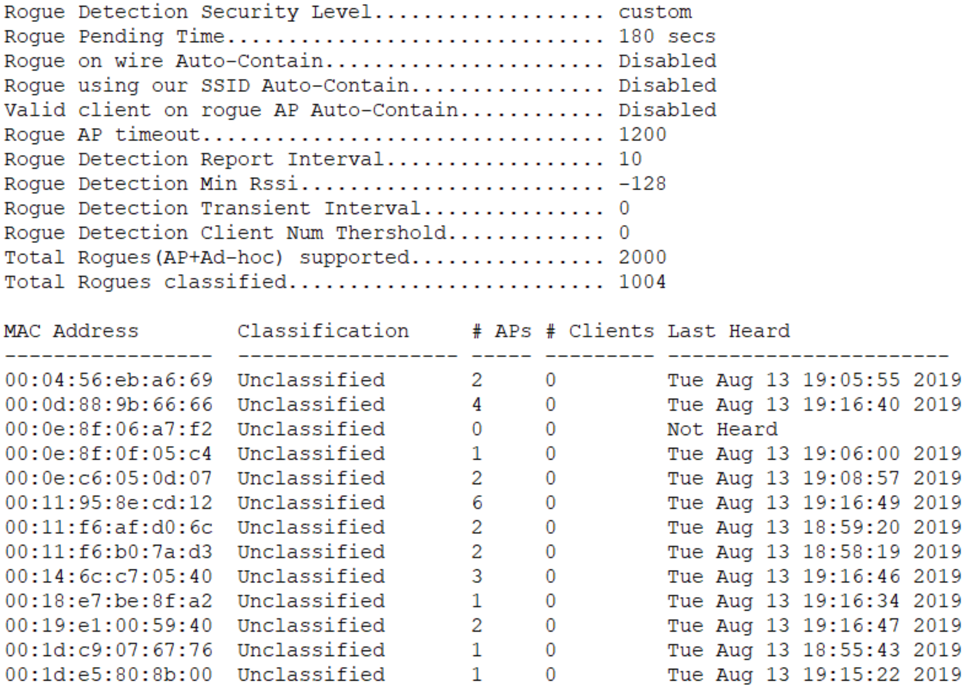
Measure the diversity– Use “config_diversity”

```
SYNTAX:
config_diversity([object 1, object 2,…,object N]) # list as input!!!


#Compare SSID configs
config_diversity([ssid1,ssid2])

Diversity metric is equal  0  %
This metric is calculated for  2  config items of type  SSID config

#Compare SSID configs
Diversity metric is equal  1  %
This metric is calculated for  2  config items of type  SSID config
```

- Hint: Get the big picture first, then dig deeper into details

Analyze rogue APs

Can we do something useful with this list?



```
Rogue Detection Security Level.................... custom
Rogue Pending Time................................ 180 secs
Rogue on wire Auto-Contain........................ Disabled
Rogue using our SSID Auto-Contain................. Disabled
Valid client on rogue AP Auto-Contain............. Disabled
Rogue AP timeout.................................. 1200
Rogue Detection Report Interval................... 10
Rogue Detection Min Rssi.......................... -128
Rogue Detection Transient Interval................ 0
Rogue Detection Client Num Thershold.............. 0
Total Rogues(AP+Ad-hoc) supported................. 2000
Total Rogues classified........................... 1004

MAC Address        Classification     # APs # Clients Last Heard
-----------------  ------------------ ----- --------- -----------------------
00:04:56:eb:a6:69  Unclassified       2     0         Tue Aug 13 19:05:55 2019
00:0d:88:9b:66:66  Unclassified       4     0         Tue Aug 13 19:16:40 2019
00:0e:8f:06:a7:f2  Unclassified       0     0         Not Heard
00:0e:8f:0f:05:c4  Unclassified       1     0         Tue Aug 13 19:06:00 2019
00:0e:c6:05:0d:07  Unclassified       2     0         Tue Aug 13 19:08:57 2019
00:11:95:8e:cd:12  Unclassified       6     0         Tue Aug 13 19:16:49 2019
00:11:f6:af:d0:6c  Unclassified       2     0         Tue Aug 13 18:59:20 2019
00:11:f6:b0:7a:d3  Unclassified       2     0         Tue Aug 13 18:58:19 2019
00:14:6c:c7:05:40  Unclassified       3     0         Tue Aug 13 19:16:46 2019
00:18:e7:be:8f:a2  Unclassified       1     0         Tue Aug 13 19:16:34 2019
00:19:e1:00:59:40  Unclassified       2     0         Tue Aug 13 19:16:47 2019
00:1d:c9:07:67:76  Unclassified       1     0         Tue Aug 13 18:55:43 2019
00:1d:e5:80:8b:00  Unclassified       1     0         Tue Aug 13 19:15:22 2019
```

Get the nice summary for all rogue APs - 1

```
SYNTAX:
Rogue_ap_summary(wlc_config)

#See how many rogue APs are dangerous ones (close to our APs)
Rogue AP summary for  WC-MSK-CISCO-CX:
The overall number of rogue APs : 1998
The number of rogue AP with highest RSSI  -10  dBm =  0
The number of rogue AP with highest RSSI  -20  dBm =  5
The number of rogue AP with highest RSSI  -30  dBm =  19
The number of rogue AP with highest RSSI  -40  dBm =  51
The number of rogue AP with highest RSSI  -50  dBm =  125
The number of rogue AP with highest RSSI  -60  dBm =  236
The number of rogue AP with highest RSSI  -70  dBm =  432
The number of rogue AP with highest RSSI  -80  dBm =  853
The number of rogue AP with highest RSSI  -90  dBm =  1989
<..output continues next page..>
```

Get the nice summary for all rogue APs - 2

```
SYNTAX:
Rogue_ap_summary(wlc_config)

#See which rogue APs has the most impact
The most impacting rogue APs in this environment:
Currently hardcoded values for impact are:
* RSSI > -50 dBm
* number of detecting APs > 3
54:4a:00:d1:fd:00  #Check these rogues first – they MIGHT have high impact
a0:93:51:38:a8:60
f0:9e:63:70:ef:b0
00:fc:ba:0b:b9:e0
<..output continues next page..>
```

Get the nice summary for all rogue APs - 3

```
SYNTAX:
Rogue_ap_summary(wlc_config)

#See which APs from our network are impacted
The impacted APs in this environment:
WA-MSK-CX-001 60 %
WA-MSK-CX-006 30 %
WA-MSK-CX-215 20 %
WA-MSK-CX-007 53 %
WA-MSK-CX-330 28 %
WA-MSK-CX-003 16 %
<..output omitted..>
#Know manufacturers of rogue APs
Most common manufacturers of rogue APs are:
[('Cisco', 199), ('Sercomm', 168), ('RuckusWi', 158), ('ZyxelCom', 127),
('Routerboard', 115)]
```

Visualize the channel utilization

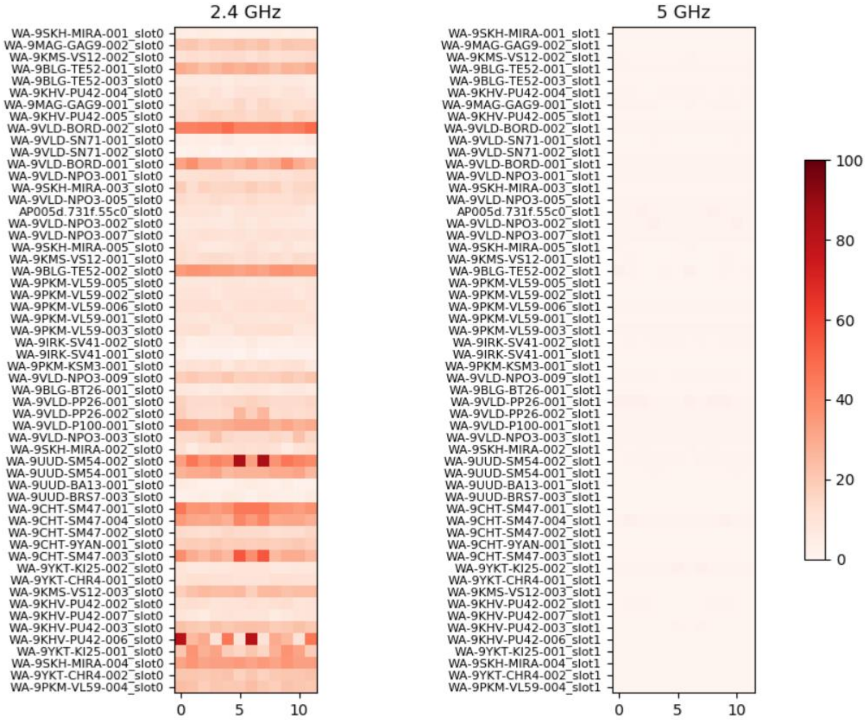
Why not to turn text data into graph?

- One picture worth a thousands of text words
- Find some patterns based on data

Examples:

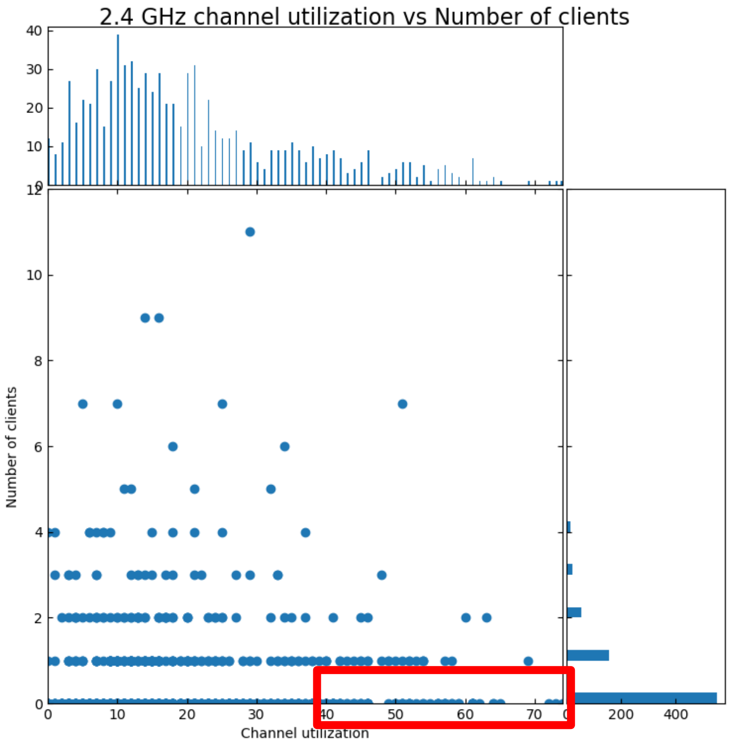
- Periodically collected WLC config -> channel utilization changes in time
- Which factors define channel utilization? -> scatterplot data for every AP in network

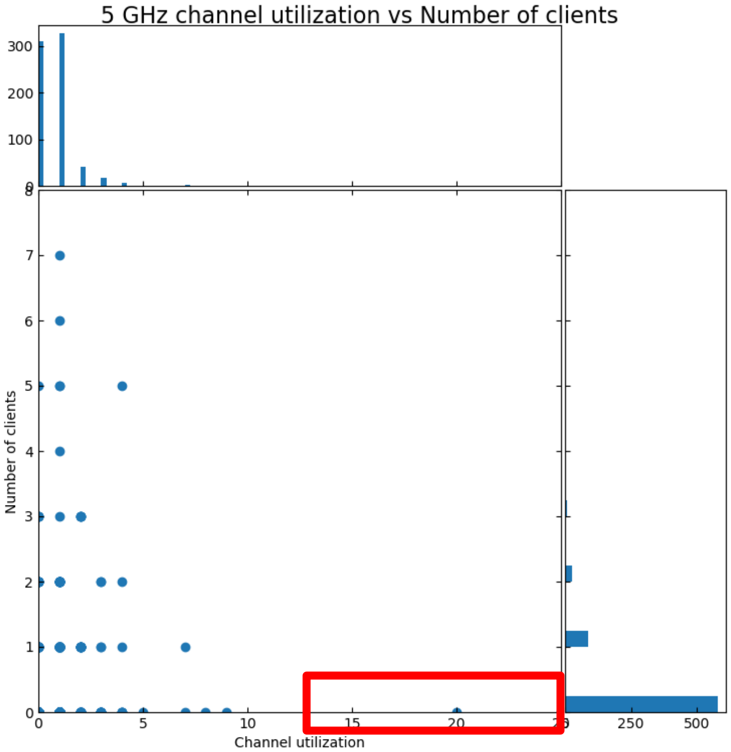
Heatmap of channel utilization in time



```
SYNTAX:
channel_utilization_visual(archive)
```

Does channel utilization depends on # of clients?

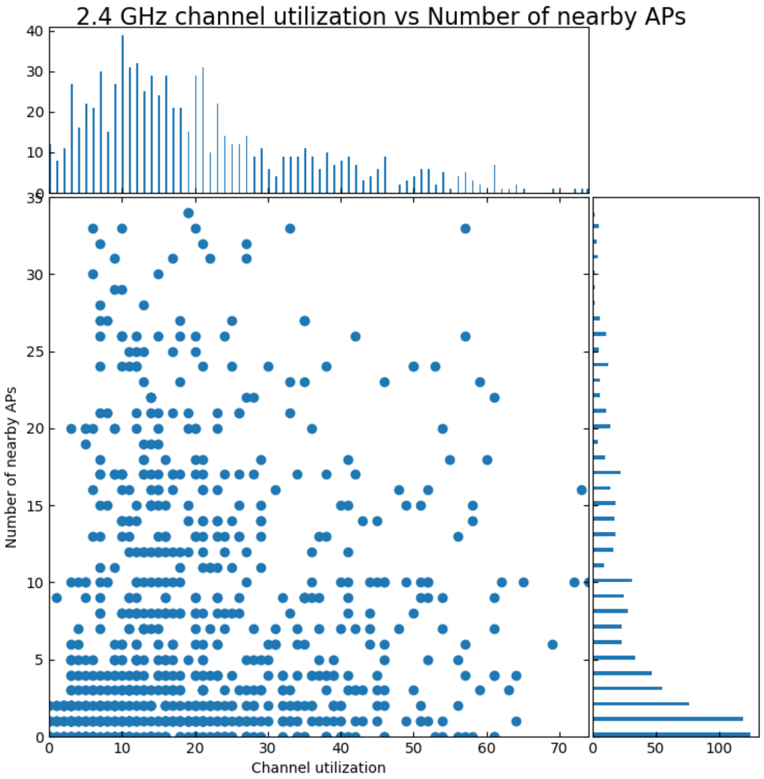


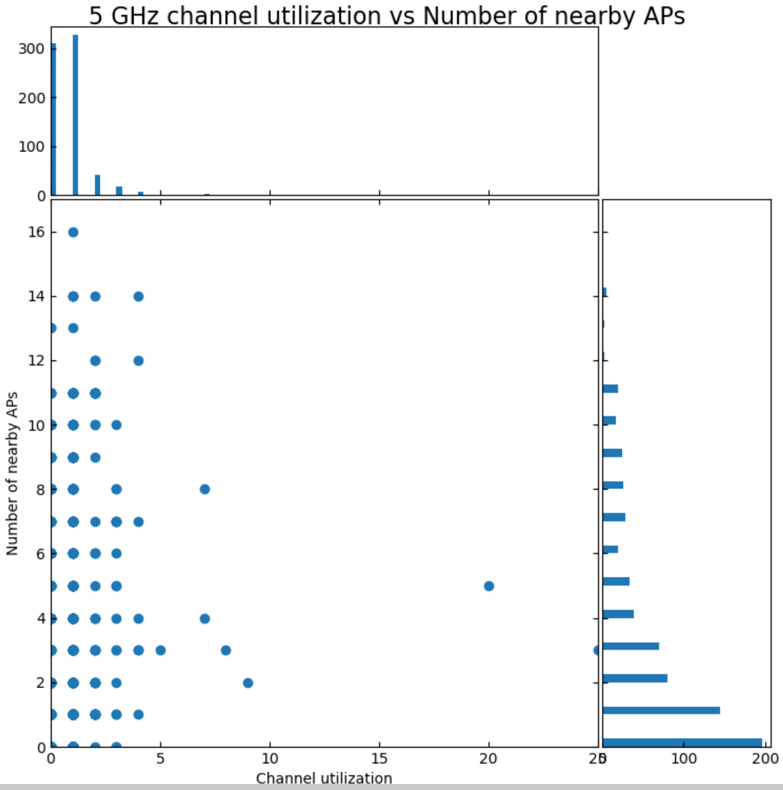


SYNTAX:

```
utilization_clients_scatterplot(archive)
```

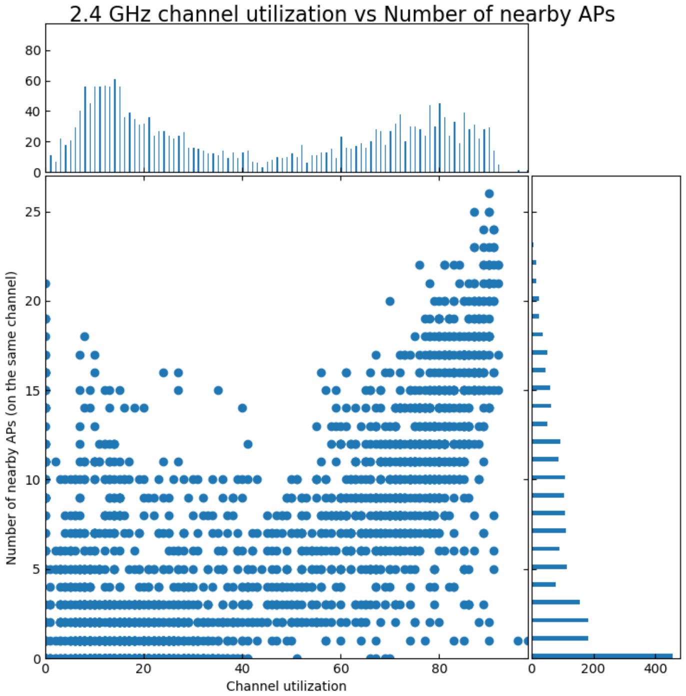
Does channel utilization depends on nearby APs?


2.4 GHz channel utilization vs Number of nearby APs


5 GHz channel utilization vs Number of nearby APs

```
SYNTAX:
utilization_nearby_aps_scatterplot(archive)
```

Does channel utilization depends on nearby APs?



```
SYNTAX:
utilization_same_channel_nearby_aps_scatterplot(archive)
```

What is next?

Next steps and call to action

- First and foremost - **Test and give feedback**
- Share ideas for the problems and tasks you face in everyday job
- Join the development

How to join?

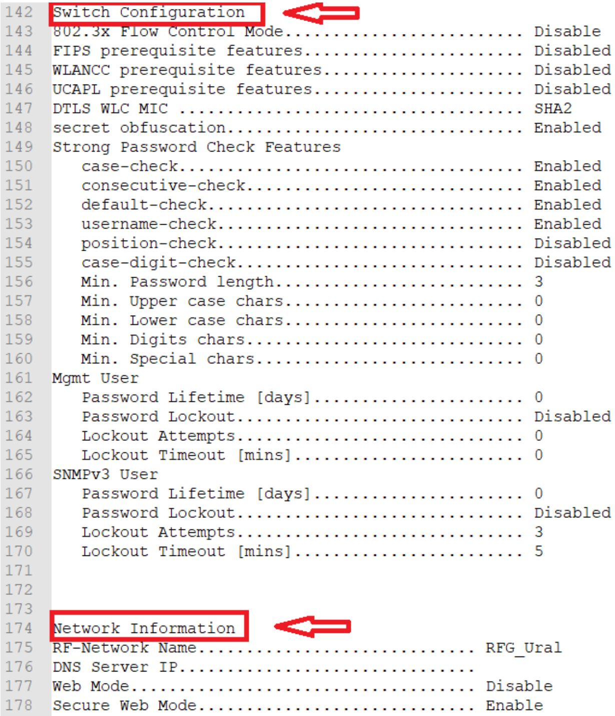
HERE IS THE link to DevNet github repo!

The approach to parsing (if you would like to co-develop)

WLC config file

- Quite large for text file (up to 50 Mbytes)
- Mix of configuration and operational data
- Structured to some extent

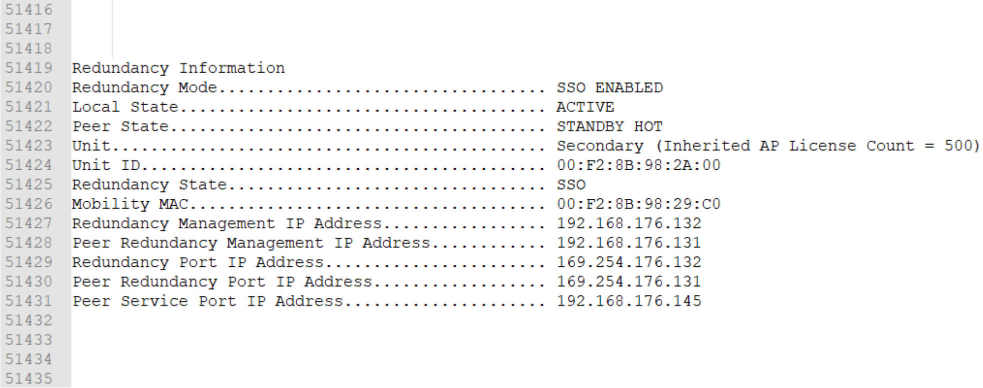
WLC config sections



```
Switch Configuration
802.3x Flow Control Mode......................... Disable
FIPS prerequisite features....................... Disabled
WLANCC prerequisite features..................... Disabled
UCAPL prerequisite features...................... Disabled
DTLS WLC MIC .................................... SHA2
secret obfuscation............................... Enabled
Strong Password Check Features
   case-check.................................... Enabled
   consecutive-check............................. Enabled
   default-check................................. Enabled
   username-check................................ Enabled
   position-check................................ Disabled
   case-digit-check.............................. Disabled
   Min. Password length.......................... 3
   Min. Upper case chars......................... 0
   Min. Lower case chars......................... 0
   Min. Digits chars............................. 0
   Min. Special chars............................ 0
Mgmt User
   Password Lifetime [days]...................... 0
   Password Lockout.............................. Disabled
   Lockout Attempts.............................. 0
   Lockout Timeout [mins]........................ 0
SNMPv3 User
   Password Lifetime [days]...................... 0
   Password Lockout.............................. Disabled
   Lockout Attempts.............................. 3
   Lockout Timeout [mins]........................ 5



Network Information
RF-Network Name............................. RFG_Ural
DNS Server IP...............................
Web Mode.................................... Disable
Secure Web Mode............................. Enable
```

- Start with easy distinguishable word sequence
- First step to parsing
- About 80 sections

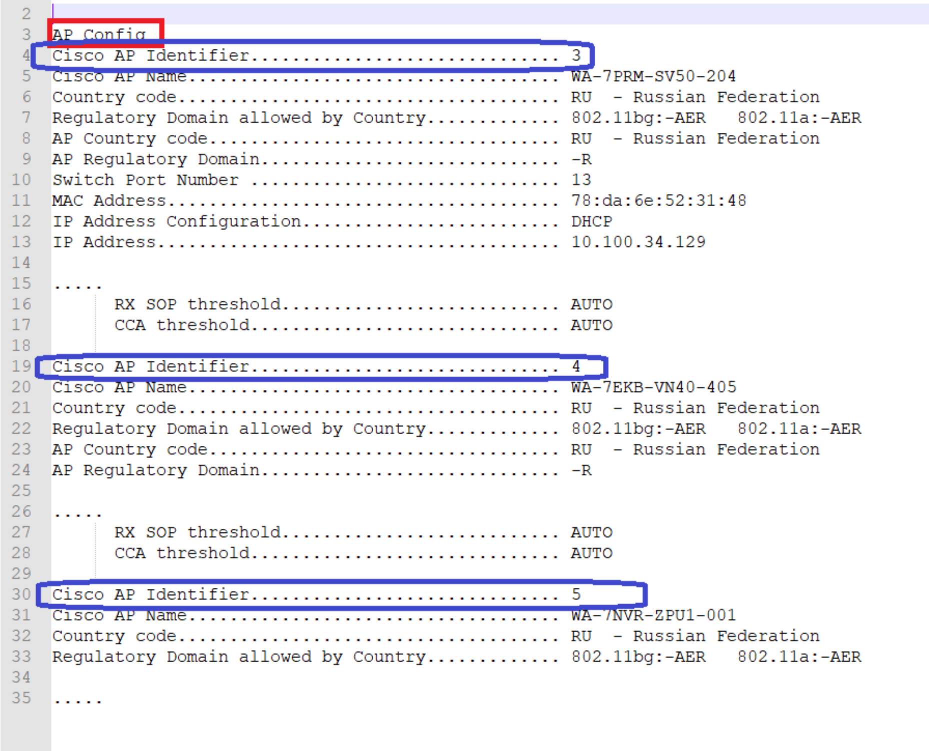
Two types of config sections – type 1



```
Redundancy Information
Redundancy Mode.................................. SSO ENABLED
Local State...................................... ACTIVE
Peer State....................................... STANDBY HOT
Unit............................................. Secondary (Inherited AP License Count = 500)
Unit ID.......................................... 00:F2:8B:98:2A:00
Redundancy State................................. SSO
Mobility MAC..................................... 00:F2:8B:98:29:C0
Redundancy Management IP Address................. 192.168.176.132
Peer Redundancy Management IP Address............ 192.168.176.131
Redundancy Port IP Address....................... 169.254.176.132
Peer Redundancy Port IP Address.................. 169.254.176.131
Peer Service Port IP Address..................... 192.168.176.145
```

- Every parameter is unique and not repeated
- Usually it is WLC (whole system) config
- The number of parameters is usually the same (may differ with sw version)

Two types of config sections – type 2

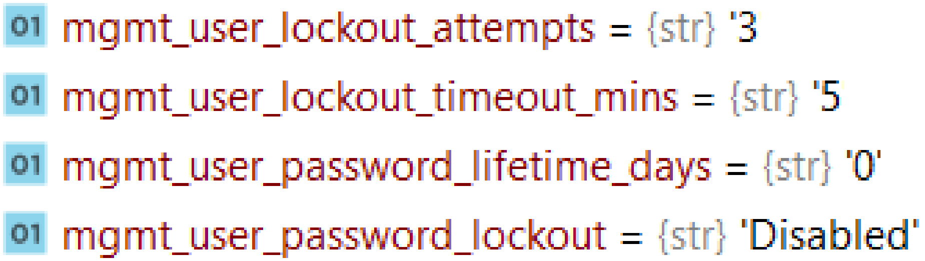


```

AP Config
Cisco AP Identifier.............................. 3
Cisco AP Name.................................... WA-7PRM-SV50-204
Country code..................................... RU  - Russian Federation
Regulatory Domain allowed by Country............. 802.11bg:-AER   802.11a:-AER
AP Country code.................................. RU  - Russian Federation
AP Regulatory Domain............................. -R
Switch Port Number .............................. 13
MAC Address...................................... 78:da:6e:52:31:48
IP Address Configuration......................... DHCP
IP Address....................................... 10.100.34.129

.....
      RX SOP threshold........................... AUTO
      CCA threshold.............................. AUTO

Cisco AP Identifier.............................. 4
Cisco AP Name.................................... WA-7EKB-VN40-405
Country code..................................... RU  - Russian Federation
Regulatory Domain allowed by Country............. 802.11bg:-AER   802.11a:-AER
AP Country code.................................. RU  - Russian Federation
AP Regulatory Domain............................. -R

.....
      RX SOP threshold........................... AUTO
      CCA threshold.............................. AUTO

Cisco AP Identifier.............................. 5
Cisco AP Name.................................... WA-7NVR-ZPU1-001
Country code..................................... RU  - Russian Federation
Regulatory Domain allowed by Country............. 802.11bg:-AER   802.11a:-AER

.....
```

- Contains the list of objects
- Objects have identifier of some sort
- The same repetitive set of parameters for every object
- The number of objects in list may differ (WLANs configured, AP associated etc.)

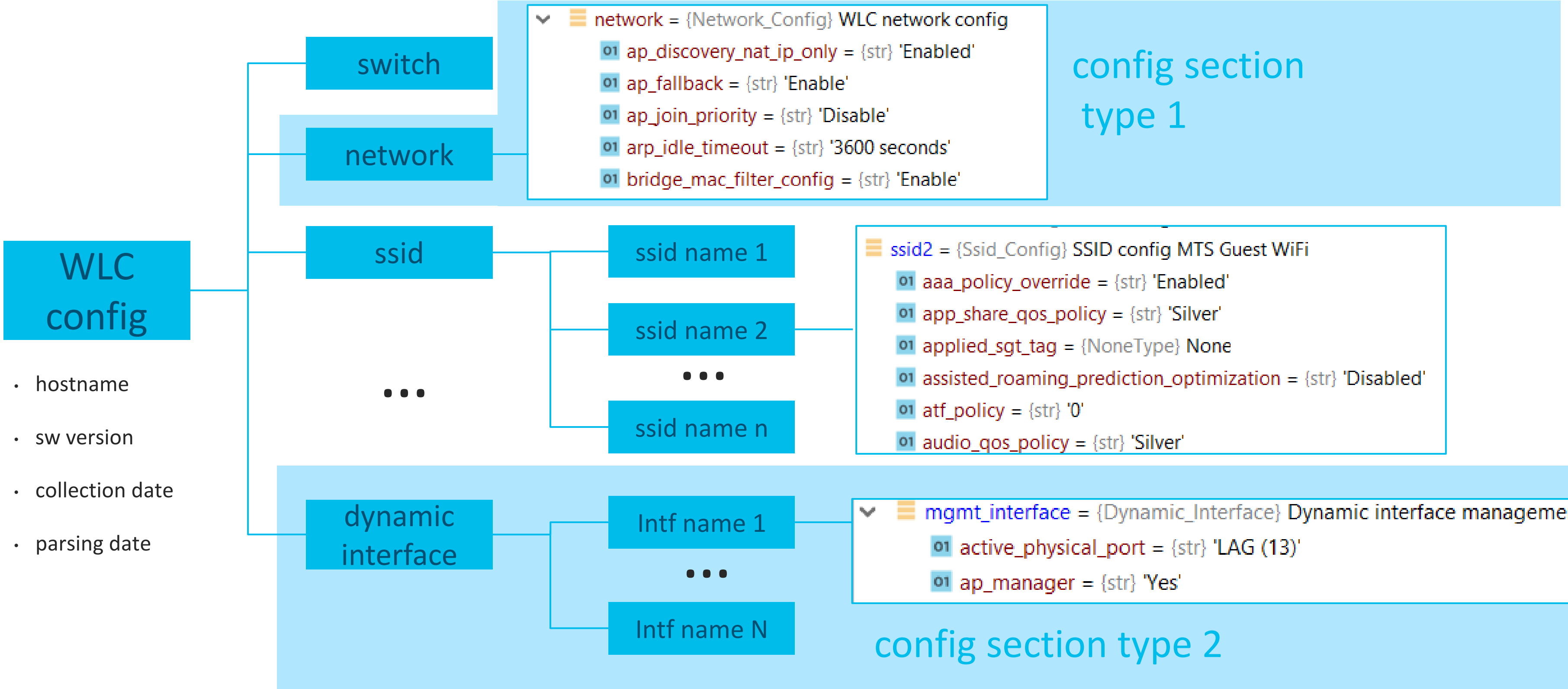
Key element of data model

- Attribute (key) = name of parameter
- Value = configuration applied
- Dictionary-like: quickly access value by name of attribute



```
mgmt_user_lockout_attempts = {str} '3'
mgmt_user_lockout_timeout_mins = {str} '5'
mgmt_user_password_lifetime_days = {str} '0'
mgmt_user_password_lockout = {str} 'Disabled'
```

The object tree for WLC config



```
network = {Network_Config} WLC network config
    ap_discovery_nat_ip_only = {str} 'Enabled'
    ap_fallback = {str} 'Enable'
    ap_join_priority = {str} 'Disable'
    arp_idle_timeout = {str} '3600 seconds'
    bridge_mac_filter_config = {str} 'Enable'
```

config section type 1

```
ssid2 = {Ssid_Config} SSID config MTS Guest WiFi
    aaa_policy_override = {str} 'Enabled'
    app_share_qos_policy = {str} 'Silver'
    applied_sgt_tag = {NoneType} None
    assisted_roaming_prediction_optimization = {str} 'Disabled'
    atf_policy = {str} '0'
    audio_qos_policy = {str} 'Silver'
```

```
mgmt_interface = {Dynamic_Interface} Dynamic interface manageme
    active_physical_port = {str} 'LAG (13)'
    ap_manager = {str} 'Yes'
```

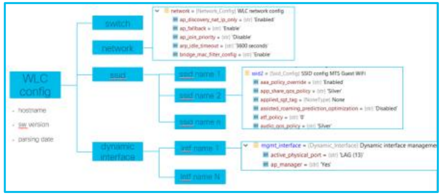
config section type 2

- hostname
- sw version
- collection date
- parsing date

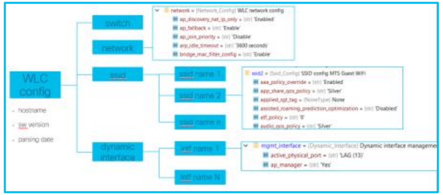
Multiple WLCs can be combined into one element

WLCs =

- WLC name 1
- WLC name 2
- WLC name N

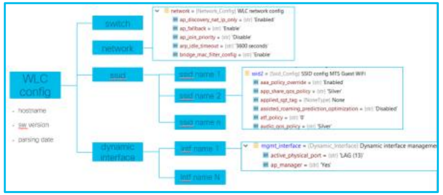


```
network = (Network_Config) WLC network config
    ap_discovery_nat_ip_only = (str) 'Enabled'
    ap_fallback = (str) 'Enable'
    ap_join_priority = (str) 'Disable'
    arp_idle_timeout = (str) '3600 seconds'
    bridge_mac_filter_config = (str) 'Enable'
ssid2 = (Ssid_Config) SSID config MTS Guest WiFi
    aaa_policy_override = (str) 'Enabled'
    app_share_qos_policy = (str) 'Silver'
    applied_sgt_tag = (NoneType) None
    assisted_roaming_prediction_optimization = (str) 'Disabled'
    atf_policy = (str) '0'
    audio_qos_policy = (str) 'Silver'
mgmt_interface = (Dynamic_Interface) Dynamic interface manageme
    active_physical_port = (str) 'LAG (13)'
    ap_manager = (str) 'Yes'
```



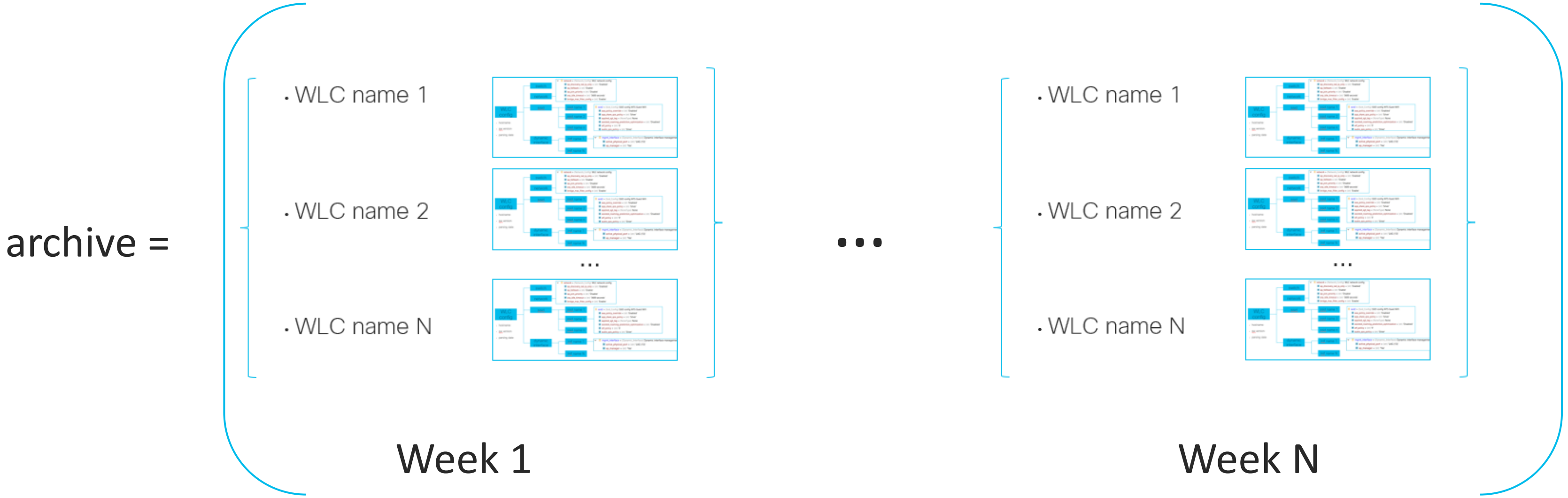
```
network = (Network_Config) WLC network config
    ap_discovery_nat_ip_only = (str) 'Enabled'
    ap_fallback = (str) 'Enable'
    ap_join_priority = (str) 'Disable'
    arp_idle_timeout = (str) '3600 seconds'
    bridge_mac_filter_config = (str) 'Enable'
ssid2 = (Ssid_Config) SSID config MTS Guest WiFi
    aaa_policy_override = (str) 'Enabled'
    app_share_qos_policy = (str) 'Silver'
    applied_sgt_tag = (NoneType) None
    assisted_roaming_prediction_optimization = (str) 'Disabled'
    atf_policy = (str) '0'
    audio_qos_policy = (str) 'Silver'
mgmt_interface = (Dynamic_Interface) Dynamic interface manageme
    active_physical_port = (str) 'LAG (13)'
    ap_manager = (str) 'Yes'
```

...



```
network = (Network_Config) WLC network config
    ap_discovery_nat_ip_only = (str) 'Enabled'
    ap_fallback = (str) 'Enable'
    ap_join_priority = (str) 'Disable'
    arp_idle_timeout = (str) '3600 seconds'
    bridge_mac_filter_config = (str) 'Enable'
ssid2 = (Ssid_Config) SSID config MTS Guest WiFi
    aaa_policy_override = (str) 'Enabled'
    app_share_qos_policy = (str) 'Silver'
    applied_sgt_tag = (NoneType) None
    assisted_roaming_prediction_optimization = (str) 'Disabled'
    atf_policy = (str) '0'
    audio_qos_policy = (str) 'Silver'
mgmt_interface = (Dynamic_Interface) Dynamic interface manageme
    active_physical_port = (str) 'LAG (13)'
    ap_manager = (str) 'Yes'
```

Multiple WLCs, multiple periodic config collection



archive =

- WLC name 1
- WLC name 2

...

- WLC name N

Week 1

...

- WLC name 1
- WLC name 2

...

- WLC name N

Week N

Overview of parsing procedure

- Separate config sections by start words
- Parse every section into Python object – two types of objects to represent two section types
- Attach the section object to WLC config object
- Repeat for every WLC in file

CISCO