Router Analytics 2.0 – Wi-Fi Events

Tego Chang
2017/11/16

NETGEAR®

RA-REQ-P2-034	Wi-Fi connect analytics
Requirements	• Wireless device connect success/fail (with reason) • Disconnect (triggered by AP or client), and reason
User Stories	Knowing the reason why connection is unsuccessful (fail case) or unstable (disconnect case). This could be the reference for improving wireless experience and compatibility.
Router SKUs	Nighthawk, Orbi

```
Placement: it’s individual object in RA dictionary

Cases:
1. Connect success
2. Connect fail
3. Disconnect by AP
4. Disconnect by client
```

NETGEAR®

Rename of Wi-Fi steering

+ Initial steering => pre-association steering
+ Active steering => post-association steering
  - Main reason
    - If we change the reason code “Active” in QCA, it might cause misunderstanding in cooperation with QCA.
  - Other pros
    - Avoid development confuse
    - Naming is more close to meaning on STA status concern.
    - More recognizable in Internet
  - Note
    - Same naming with QCA but QCA has a trigger condition of channel overloading while NTGR doesn’t.

NETGEAR®

RA-REQ-P2-021 & RA-REQ-P2-032	Wi-Fi steer analytics
Requirements	• Wireless device pre-association steering & post-association steering success/fail (with reason) • Decisions regarding steer to 2.4G/5G, or steer to base/satellite (AP steering)
User Stories	Knowing the decision differences on band and AP steering between client device and router, which could be a reference to design algorithm to optimize network capability.
Router SKUs	Nighthawk (only band steering), Orbi

Steering type	**Definition**
Pre-association steering	The steer decision is generated before client connects to AP. (Pre-connect; on steering, the client haven't connected to AP.)
Post-association steering	The steering decision is generated after client connects to AP. (Post-connect; on steering, the client already connected to AP.)

```
Placement: it's individual object in RA dictionary

Cases:
1. Pre-association band steering (success/fail to 2.4G/5G)
2. Pre-association AP steering (success/fail to which AP)
3. Post-association band steering (success/fail to 2.4G/5G)
4. Post-association AP steering (success/fail to which AP)
```

NETGEAR®

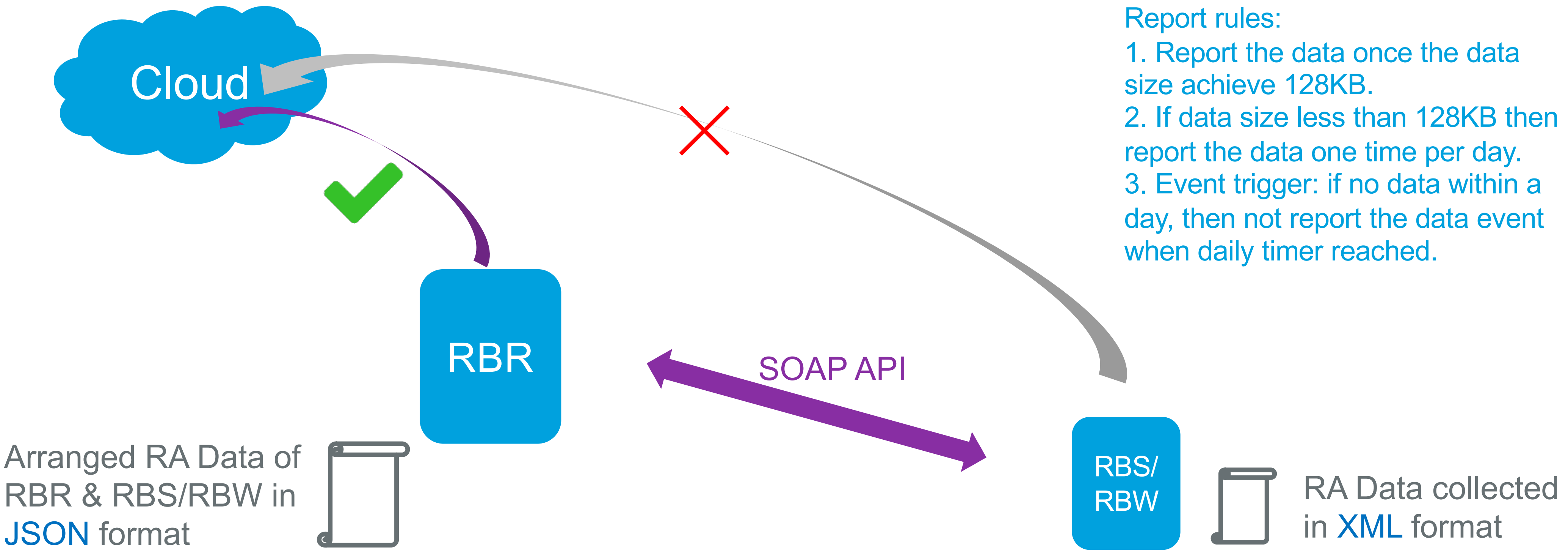
RA Data Flow

NETGEAR®

Orbi Wi-Fi Analytics Data Collection

Limitation:
The last second to initiate steer in the first round 5 min, and complete in the second round 5 min's first second, then this event will be dropped.

Report rules:
1. Report the data once the data size achieve 128KB.
2. If data size less than 128KB then report the data one time per day.
3. Event trigger: if no data within a day, then not report the data event when daily timer reached.



Arranged RA Data of RBR & RBS/RBW in JSON format

RA Data collected in XML format

1. Base/satellite collect period = 300 second
2. Base & sate sync period = 300 s
~~3. Base report period = 300 s~~ (but report only when there's event => event-triggered)

NETGEAR

SOAP API Format

- for Wi-Fi RA between RBR and RBS/RBW

Function name	Input		Response	
	Argument	Format/Value	Parameter	XML Format/Value
GetWifiConnectEvent Service Type: DeviceInfo			Accumulated Wi-Fi connect/steer events since last collection	This output parameter shall be arranged in array structure: <AccumulatedWifiConnectEvent> <numberOfEvents>string</numberOfEvents> <Event> <beginTimeStamp>string</beginTimeStamp> <endTimeStamp>string</endTimeStamp> <eventType>string</eventType> <Reason>string</Reason> <steerType>string</steerType> <steerReason>string</steerReason> <steerSuggestion>string</steerSuggestion> <steerSuggestionAPMAC>string</steerSuggestionAPMAC> <clientMAC>string</clientMAC> <connBandChannel>string</connBandChannel> <RSSI>string</RSSI> <deviceType>string</deviceType> <deviceName>string</deviceName> <apModel>string</apModel> <apMAC>string</apMAC> <apName>string</apName> </Event> … <Event> … </Event> </AccumulatedWifiConnectEvent>

Design of RA Data Report Format - JSON

JSON Format Brief

"routerHardware": {			
"productFamily":	"wifiSystem",	String	[“router” | “wifiSystem” | “rangeExtender” | “dsl” | “cable”]
"modelName":	"rbr50",	String	
"stage":	"dev",	String	["prod" | "beta" | "dev"]
"deviceInfo": {			
"macAddress":	"201407112a21",	String	
"serialNumber":	"9AE2222DBFE2A"	String	
},			
"eventType":	3,	Integer	• 0 – Kernel Reboot • 1 - Daily Updates • 2 – Unsolicited Events (High CPU / Memory Utilizations) • 3 – WiFi Parameters • 4 – User Options & Firmware Version #
"timeStamp": {			
"date":	[2016, 10, 28],	Array of Integers	
"time":	[9, 31, 29]	Array of Integers	
}			
},			
"firmware": {			
"fwVersion":	"V1.0.8.22",		
"fwLastUpdate":	"",		
"fwLastChecked":	"2017_4.10_22:35:0"		
},			
"autoUpdateOption":	1,		
"routerAnalyticsOption":	1		
}			

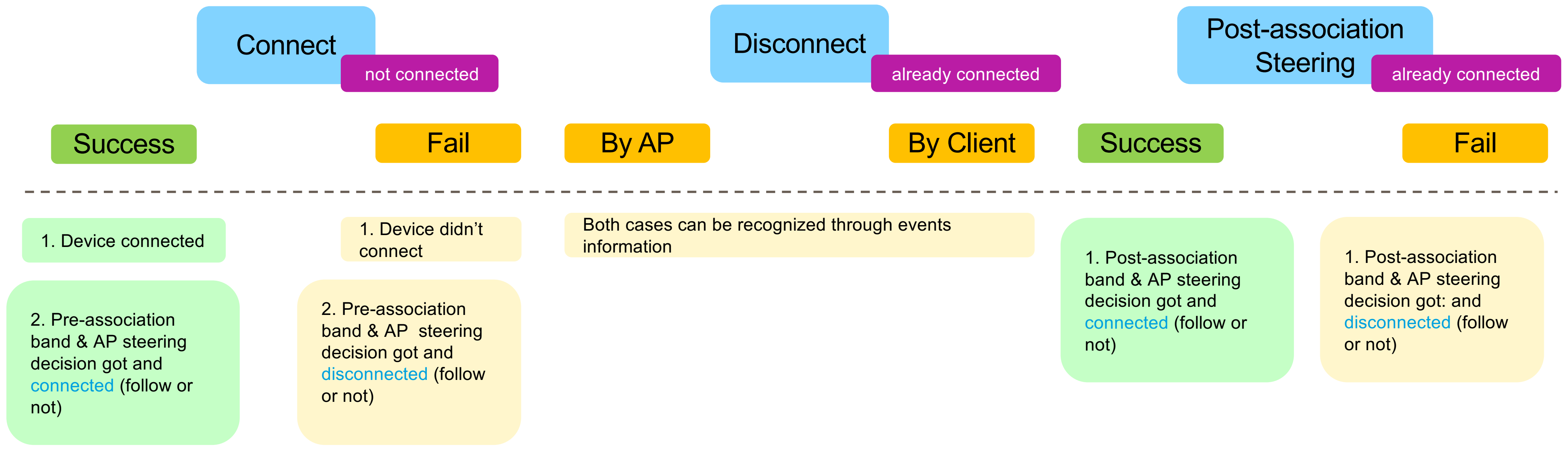
Goal and design concept

+ The primary goal is to evaluate the Wi-Fi quality of our product
  - Connect success/fail rate, including initial & active steering
+ Then, improve the quality
  - The fail and disconnect cases with reason analysis
+ Finally, investigate the decisions of steering
  - Steer decision follow or not tracking

Note:

1. The primary concept is to evaluate our Wi-Fi quality through connect success/fail; once enhanced, then will be the decision accuracy of steering.
2. For both connect & steer cases, the basic duty of AP is to make sure client can connect to any AP.
3. The disconnect & reconnect process during active steering shall **NOT** be removed while reporting to the cloud.

Design Architecture



1. Device connected

2. Pre-association band & AP steering decision got and connected (follow or not)

1. Device didn't connect

2. Pre-association band & AP steering decision got and disconnected (follow or not)

Both cases can be recognized through events information

1. Post-association band & AP steering decision got and connected (follow or not)

1. Post-association band & AP steering decision got: and disconnected (follow or not)

Note:

1. In the success case 2 & 3 (initial steering), follow or not shall be known by AP through event analysis => Yes
2. QCA's blacklist is assumed to have timeout mechanism? => Yes
3. QCA's AP steering is assumed to have no timeout mechanism as smart connect? =>Yes, it has black list timeout mechanism, but with different timer
4. On initial band steering:

(1) QCA: enables only when channel overloading, comes with a blacklist of channel/band; while channel available, client is free to decide which band to connect.

NETGEAR®

On eventType = 1 ~ 4

RA-REQ-P2-021, 032, and 034

Wi-Fi connect analytics

“wifiConnectEvent”:{			
“numberOfEvents”:	14,	Integer	
“event”:[{			
“beginTimeStamp”	“2016-10-27 09:06:13”,	String	yyyy-MM-dd HH:mm:ss
“endTimeStamp”	“”,	String	yyyy-MM-dd HH:mm:ss detail: 1. If eventType= “1:Connect success” or “2:Connect fail” or “3:Disconnect by ap” or “4:Disconnect by client”, then the endTimeStamp format will be NULL or blank.
“eventType”:	“Connect success”,	String	1. Connect success (includes pre-association steer success) 2. Connect fail (includes pre-association steer fail) 3. Disconnect by ap 4. Disconnect by client 5. Post-association steer success (the device’s behavior is the same as AP’s suggestion) 6. Post-association steer fail
“reason”:	“”,	String	Template: code#:description 1. Connect => #:either <reason> or <status> of the last event 2. Disconnect => #:<reason> 3. Steer => #:Steer result <result>
“steerType”	“2:BTM and Blacklist”,	String	#:Steer start <type>
“steerReason”	“4:IdleOffload”,	String	#:Steer start <reason>
“steerSuggestion”:	“Steer to 2.4G”,	String	1. Steer to 2.4G 2. Steer to 5G 3. Steer to ap: mac address (the information comes from BTM request)
“steerSuggestionAPMAC”:	"64:20:0C:95:F1:AF",	String	
“clientMAC”:	“64:20:0C:95:F9:AF”,	String	
“connBandChannel”:	"2.4G:6",	String	
“RSSI”	“-56dBm”	String	
“deviceType”:	“iphone6”,	String	
“deviceName”:	“netgearphone”,	String	apMAC & apName can be acquired (Zefu), through driver event information parsing

Example1: Pre-association steer success

“beginTimeStamp": "2017-12-08 03:26:34",
“endTimeStamp": "",
"eventType": "connect success",
"reason": "0:Success",
"steerType": "6:Pre-Association",
"steerReason": "", --- There is no steer reason code in Pre-Associate Case, only reason is Band Overloading
"steerSuggestion": "", --- Empty, because client didn’t connected on AP before steer, No SteerSuggestion AP so no steerSuggestion.
"steerSuggestionAPMAC": "", --- Empty, because client didn’t connected on AP before steer, No SteerSuggestion AP.
"clientMAC": "AC:22:0B:BE:41:7C",
"connBandChannel": "5G:48",
"rssi": "-58dbm",
"deviceType": "", --- If client first time connect to AP, no deviceType data in attach device list
"deviceName": "", --- If client first time connect to AP, no deviceName data in attach device list
"apModel": "RBR40",
"apMAC": "b0:b9:8a:5e:e5:d4",
"apName": "Daniel_test"

Example2: Pre-association steer fail

```
“beginTimeStamp": "2017-12-08 03:26:34",
“endTimeStamp": "",
"reason": "7:AssocTimeout",
"steerType": "6:Pre-Association",
"steerReason": "", --- Same as success case
"steerSuggestion": "", --- Same as success case
"steerSuggestionAPMAC": "",
"clientMAC": "1C:7B:21:56:C2:84",
"connBandChannel": "",
"rssi": "-64dbm",
"deviceType": "", --- Same as success case
"deviceName": "", --- Same as success case
"apModel": "RBR40",
"apMAC": "", --- Client didn’t connect to AP
"apName": "Daniel_test"
```

NETGEAR®

On eventType = 5 or 6

RA-REQ-P2-021, 032, and 034

Wi-Fi connect analytics

“wifiConnectEvent”:{			
“numberOfEvents”:	14,	Integer	
“event”:[{			
“beginTimeStamp”	“2016-10-27 09:06:13”,	String	timestamp of steer start
“endTimeStamp”	“2016-10-27 09:06:15”,	String	timestamp of steer result
“eventType”:	“Post-association steer success”,	String	1. Connect success (includes initial steer success) 2. Connect fail (includes initial steer fail) 3. Disconnect by ap 4. Disconnect by client 5. Post-association steer success (the device’s behavior is the same as AP’s suggestion) 6. Post-association steer fail
“reason”:	“0:Success”,	String	Template: code:description 1. Connect => #:either <reason> or <status> of the last event 2. Disconnect => #:<reason> 3. Steer => #:Steer result <result>
“steerType”	“2:BTM and Blacklist”,	String	#:Steer start <type>
“steerReason”	“4:IdleOffload”,	String	#:Steer start <reason>
“steerSuggestion”:	“Steer to 2.4G”,	String	1. Steer to 2.4G 2. Steer to 5G 3. Steer to ap: mac address (the information comes from BTM request)
“steerSuggestionAPMAC”:	"64:20:0C:95:F1:AF",	String	
“clientMAC”:	“64:20:0C:95:F9:AF”,	String	
“connBandChannel”:	"2.4G:6",	String	
“RSSI”	“-56dBm”	String	
“deviceType”:	“iphone6”,	String	
“deviceName”:	“netgearphone”,	String	
“apModel”	“rbr”	String	1. To distinguish base or satellite, but can’t not know the exact sku e.g. desktop or mini 2. Could also be other brand name router or unknown
“apMAC”:	"64:20:0C:95:F1:AF"	String	This is the router connected through association event from connect

Object Collection When

+ On collecting when:
  - Periodically collecting => 300 seconds
  - However, once syslog volume reach certain threshold (512 kB), it then start again.
    - This item required to check with ODM if syslog or user space buffer has size limitation. => It's adjustable (default size Obi will check).

+ On burst traffic situation:
  - The original idea is to have burst attempts mechanism to consider case that, e.g. once same mac with same eventType exceed certain threshold (frequency, e.g. number/time), don't do collecting to RAE to avoid JSON file size too huge.
    - However, the duplicate events could still mean something, e.g. client retry mechanism before reaching ap's band steer timeout.
    - Therefore, the idea is changed to collect all events but with appropriate transmission parameters set.
  - Conditions:
    - Size limitation:
      - On each RA data transmission from satellite to base, e.g. transmit 100 events at most in one forwarding. (value setting is still TBD)
      - On each RA report from base to S3 bucket (mechanism not implemented yet, await internal beta result.)
    - Time limitation:
      - Satellite to discard the existing collected data if base doesn't query after reaching SOAP timeout.
      - Base doesn't wait for satellite's response of collected data if timeout is reached.
    - Discard mechanism shall be considered to avoid attacker, e.g. 100 event/second (wait for beta test, or see if general situation it value is at what level) (value setting is still TBD)

Note: SOAP limitation define shall not exceed 4K could be a reference for 100

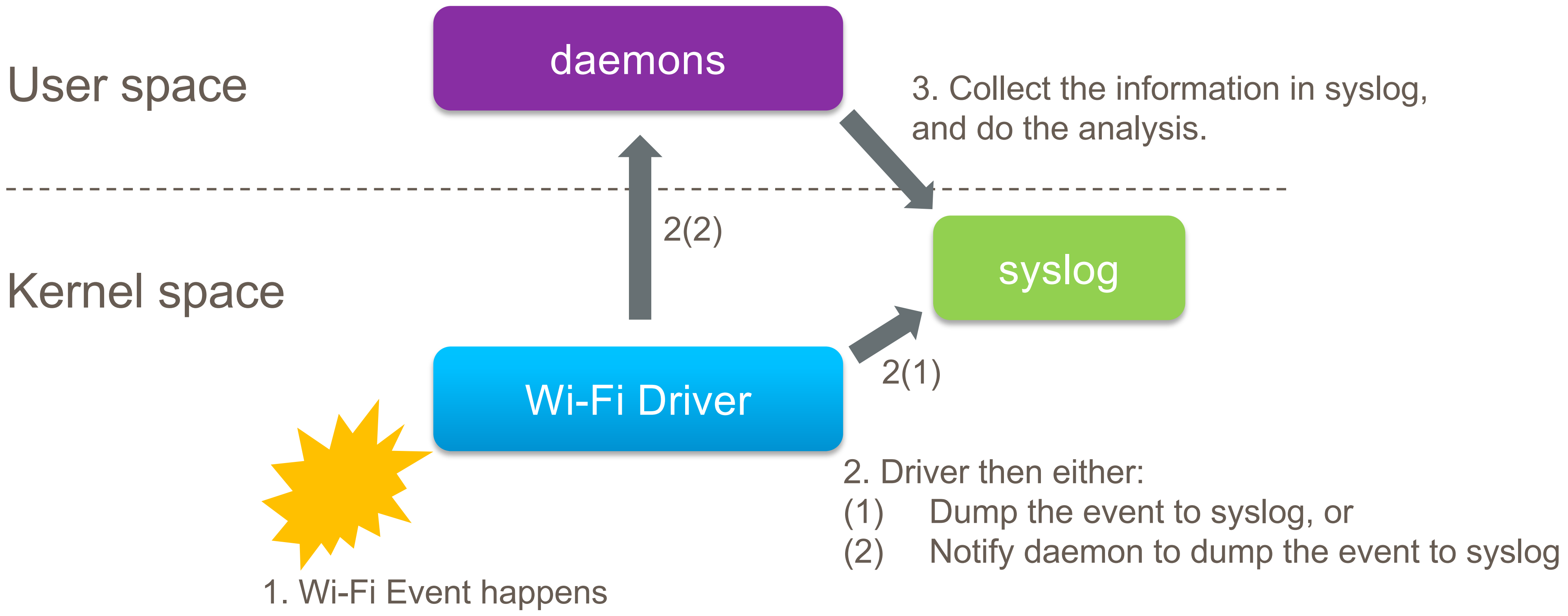
Additional Engineer Requirements

+ Periodical publish rule
  - Daily publish:
  - 1. System bootup generate a random number of secs(1~79000).
  - 2. After random number secs then start send message.
  - 3. The general report period to the cloud is 300 s.
+ When NTP fail
  - If system can't connect NTP server, system should use firmware compile time.
  - When Orbi connect to AWS server then found system time expire. It should get date from certificates and set that date in the system and resent again.
+ Retry
  - When MQTT publish fail.
    - Immediate Retry Phase: MQTT client should delay 20 secs then retry until three time fail.
    - After one hour retry Phase : If still send fail, after one hour MQTT client should resent (Also keep Immediate Retry function.)
    - If one hour retry still fail that not need retry. Just wait to next day's publish
+ MQTT Client ID
  - MQTT Clinet id need use router MAC address, can't use same client id

NETGEAR®

Implementation

Implementation - Event-triggered Architecture


User space
Kernel space
daemons
syslog
Wi-Fi Driver
2(2)
2(1)
3. Collect the information in syslog, and do the analysis.
2. Driver then either:
(1) Dump the event to syslog, or
(2) Notify daemon to dump the event to syslog
1. Wi-Fi Event happens

Connect Message Flow

Wireless Reason and Status Code.pdf

✓ **Connect Success**

Authentication request (received from sta) -> Authentication response (sent by AP) -> Association request (received from sta) -> Association response (sent by AP) -> Authorization (a flag from 0 to 1 in driver)

Connect Fail (Authorization flag equals to 0)

1. Authentication request (received from sta) -> Authentication response (sent by AP with a non-zero status code)

2. Authentication request (received from sta) -> Authentication response (sent by AP) -> Association request (received from sta) -> Association response (sent by AP with a non-zero status code)

3. Authentication request (received from sta) -> Authentication response (sent by AP) -> De-authentication (sent by AP or received from sta with a reason code)

4. Authentication request (received from sta) -> Authentication response (sent by AP) -> Association request (received from sta) -> Association response (sent by AP) -> De-authentication (sent by AP or received from sta with a reason code)

5. Authentication request (received from sta) -> Authentication response (sent by AP) -> Association request (received from sta) -> Association response (sent by AP) -> Dis-association (sent by AP or received from sta with a reason code)

NETGEAR®

Disconnect Message Flow

✓ **Disconnect by AP** (Authorization flag equals to 1)
1. De-authentication (sent by AP with a reason code)
2. Dis-association (sent by AP with a reason code)

Disconnect by sta (Authorization flag equals to 1)
1. De-authentication (received from sta with a reason code)
2. Dis-association (received from sta with a reason code)

NETGEAR®

Steer Message Flow – Pre-association Steering

Initial Steering success (Band, AP)
steer start (update black list) > check if association timeout > association (driver) > association (daemon, steer, information focus on client) > steer result (success=0)

✓ **Initial Steering fail** (Band, AP)
steer start (update black list) > association timeout(daemon) or association reject too many times(driver) > steer result (success=!0)

Microsoft Excel
□□ □

Note: in the case of initial steering:

1. type must be steer start with type code 7
2. reason could be acquired, but details is still TBC by ODM
3. result could partial use the DB of steer result (how many is TBC by ODM)

NETGEAR®

Steer Message Flow – Active Steering

Active Steering success (Band, AP)
steer start (update black list) > BTM request (802.11v) > BTM response > disassociation (driver, wi-fi connect) > disassociate (steer) > check if association timeout > association (driver) > association (daemon, steer, information focus on client) > steer result (success=0)

Active Steering fail (Band, AP)
1. Fail before disconnect (BTMReject, BTMResponseTimeout, PrepareFail)
   (1) BTMReject (end deivce decide not to tell which AP to steer to)
   steer start (update black list) > BTM request (802.11v) > BTM response with Reject code(802.11v) > steer result (success=!0)

   (2) BTMResponseTimeout (end device not respond)
   steer start (update black list) > BTM request (802.11v) > BTM response timeout > steer result (success=!0)

   (3) PrepareFail (BTM request stuck in driver and can't send out)
   steer start (update black list) > Send BTM request fail (802.11v) > steer result (success=!0)

2. Fail after disconnection (AuthReject, LowRSSI, ChangeTarget, AssocTimeout, ChannelChange, Unexpected BSS)
   Steer Start --> Disassociation --> Steer Result(!0)
   ✓ (1) AuthReject (end device try to connect to which is in the black list)
   steer start (update black list) > BTM request (802.11v) > BTM response > disassociation (driver, wi-fi connect) > disassociate (steer) > check if association timeout > association Reject too many times (driver) > steer result (success=!0)

   (2) LowRSSI
   A driver event trigger callback function, can abort at any time during steer process.

   (3) ChangeTarget (during steering process, the target has been changed, so change to new target)
   Special Case, check if any steer on progress at prepare steer stage. If yes and target is different, abort previous one and catch up current one.
   Is steer start (update black list)? > Check if any steer on progress, if yes, abort previous one > steer result (success=!0, Change Target) for previous one >BTM request (802.11v) > BTM response > disassociation (driver, wi-fi connect) > disassociate (steer) > check if association timeout > association (driver) > steer result (success=0)

   (4) AssocTimeout (end device can connect after the BTM > disconnect process)
   steer start (update black list) > BTM request (802.11v) > BTM response > disassociation (driver, wi-fi connect) > disassociate (steer) > check if association timeout > association timeout > steer result (success=0)

   (5) ChannelChange (front haul channel changed, so current steering process are aborted)
   A driver event trigger callback function, can abort at any time during steer process.

   (6) Unexpected BSS (end device connect to the wrong SSID, e.g. main network to guest network)
   steer start (update black list) > BTM request (802.11v) > BTM response > disassociation (driver, wi-fi connect) > disassociate (steer) > check if association timeout > association (driver) > association (daemon, steer, information focus on client) > associate on wrong BSS > steer result (success = !0)

Note:
the active steer fail reasons are all in steer result <result code>

NETGEAR®

Wi-Fi Connect

+ Modifications of W-Fi driver, adding printk(), to put the events to syslog
  - The modifications shall be considered safe although it's in driver.

<User space daemon to do the event recognition>

Event type	Judgement condition	Reason code
Connect success	If get the "AUTHORIZATION" events	NA
Connect fail	1. If the "authorization" = 0 when AP sent/received the "Dis-Association" or "De-Authentication" events 2. If the "authorization" = 0 when AP sent/received the "Association" or "Authentication" events with status code !=0	V
Disconnect by AP	The "authorization" = 1, and AP sent the "Dis-Association" or "De-Authentication" events	V
Disconnect by client	The "authorization" = 1, and AP received the "Dis-Association" or "De-Authentication" events	V

Note: Gary to implement the judgement in driver

Wi-Fi Steer

+ Modifications in the user space - lbd
  - lbd's library - band steering and part of AP steering
    - lbd doesn't support ap steering, but can know event messages such as: steer start, steer result, etc through code tracing.
  - hyd binary – in charge of overall steering
    - Algorithm can't be found in hyd
    - hyd has included lbd
  - Once an event happens, driver will notify lbd to dump the event to syslog with timestamp recorded.

<Definition of steering>

Steering type	Definition	Judgement
Pre-association steering	The steer decision is generated before client connects to AP. (Pre-connect; on steering, the client haven't connected to AP.)	Steer start event with type = 6 (Pre-Association)
Post-association steering	The steering decision is generated after client connects to AP. (Post-connect; on steering, the client already connected to AP.)	Steer start event with type != 6

Note:

1. According to the definition above, the "age" parameter is not required to use to distinguish initial and active steering.

NETGEAR®

Wi-Fi Steer

<User space daemon to do the event recognition>

Event type	Judgement condition	Reason code	Result code
Pre-association steering to 2.4G	User space wifison to decide to which band Legacy: black list with AP tx disassociation	V	V
Pre-association steering to 5G			
Pre-association steering to base	According to information defined in 802.11k & 802.11v; The result of steer to base or satellite shall be acquired by the information in BTM request/response message exchange	V	V
Pre-association steering to satellite			
Post-association steering to 2.4G	User space wifison to decide to which band Legacy: black list with AP tx disassociation	V	V
Post-association steering steering to 5G			
Post-association steering steering to base	According to information defined in 802.11k & 802.11v; The result of steer to base or satellite shall be acquired by the information in BTM request/response message exchange	V	V
Post-association steering steering to satellite			

Note:

1. In the BTM flow, 3 information shall be able to obtain, which are candidate target mac, requested target mac, and final connected target mac.

wifiConnectEvent – Rest Attributes

+ clientMAC, connBandChannel, apMAC and apName
  - All the information can be obtained through event recognition process

+ deviceType & deviceName
  - Refer to implementation of “Attached Device Table”

Note:
The polling period of Attached Device Table is 15 minutes, which might result in the first connected device’s type/name still not found, which is acceptable. We only do our best, but upon re-connecting to AP, the same device’s type/name shall be recognized this time.

RSSI mechanism

RSSI table

Mac	speed	rssi	noise	
84:78:8b:8b:10:bc	108Mbps	31	-105	newest
00:ec:0a:c9:4c:fc	433Mbps	36	-105	
f8:23:b2:8a:27:d2	390Mbps	39	-105	
ac:fd:ce:74:42:12	216Mbps	24	-105	
54:35:30:17:9f:85	26Mbps	27	-93	
38:a4:ed:a6:cb:42	72Mbps	37	-93	oldest (the last line is the oldest log)

Every 10 seconds

Step 1

```
    0s----------------------------10s--------------------------
     |          <collecting log>          | collect rssi from driver and add to
rssi table
     |                                    |
     |                                    |
```

Mac	speed	rssi	noise
84:78:8b:8b:10:bc	108Mbps	66	-105
00:ec:0a:c9:4c:fc	433Mbps	55	-105
f8:23:b2:8a:27:d2	390Mbps	44	-105
ac:fd:ce:74:42:12	216Mbps	33	-105
54:35:30:17:9f:85	26Mbps	22	-93
38:a4:ed:a6:cb:42	72Mbps	11	-93
84:78:8b:8b:10:bc	108Mbps	31	-105
00:ec:0a:c9:4c:fc	433Mbps	36	-105
f8:23:b2:8a:27:d2	390Mbps	39	-105
ac:fd:ce:74:42:12	216Mbps	24	-105
54:35:30:17:9f:85	26Mbps	27	-93

Step 2

```
    0s----------------------------10s--------------------------
     |          <collecting log>          | collect rssi from driver and add to rssi table
     |                                    | unique this table
     |                                    |
```

Mac	speed	rssi	noise
84:78:8b:8b:10:bc	108Mbps	66	-105
00:ec:0a:c9:4c:fc	433Mbps	55	-105
f8:23:b2:8a:27:d2	390Mbps	44	-105
ac:fd:ce:74:42:12	216Mbps	33	-105
54:35:30:17:9f:85	26Mbps	22	-93
38:a4:ed:a6:cb:42	72Mbps	11	-93
~~84:78:8b:8b:10:bc~~	~~108Mbps~~	~~31~~	~~-105~~
~~00:ec:0a:c9:4c:fc~~	~~433Mbps~~	~~36~~	~~-105~~
~~f8:23:b2:8a:27:d2~~	~~390Mbps~~	~~39~~	~~-105~~
~~ac:fd:ce:74:42:12~~	~~216Mbps~~	~~24~~	~~-105~~
~~54:35:30:17:9f:85~~	~~26Mbps~~	~~27~~	~~-93~~

Step 3

```
    0s----------------------------10s-------------------------->
     |          <collecting log>          | collect rssi from driver and add to
rssi table
     |                                    | unique this table
     |                                    | write rssi to the log that this 10s
collected
```

Mac	speed	rssi	noise
84:78:8b:8b:10:bc	108Mbps	66	-105
00:ec:0a:c9:4c:fc	433Mbps	55	-105
f8:23:b2:8a:27:d2	390Mbps	44	-105
ac:fd:ce:74:42:12	216Mbps	33	-105
54:35:30:17:9f:85	26Mbps	22	-93
38:a4:ed:a6:cb:42	72Mbps	11	-93

Every 5 minutes (report period)

```
    0s----------------------------10s--------------------------
     |                                    | delete the log of unconnected Mac
     |          <collecting log>          |
     |                                    |
     |                                    |
```

Mac	speed	rssi	noise	
84:78:8b:8b:10:bc	108Mbps	31	-105	newest
00:ec:0a:c9:4c:fc	433Mbps	36	-105	
f8:23:b2:8a:27:d2	390Mbps	39	-105	
ac:fd:ce:74:42:12	216Mbps	24	-105	
54:35:30:17:9f:85	26Mbps	27	-93	
~~38:a4:ed:a6:cb:42~~	~~72Mbps~~	~~37~~	~~-93~~	

Min error: event happen at 9.9 s => event happened right before the RSSI polling

Max error: event happen 0.1 (or 10.1s) => event happened right after the RSSI polling