Fing

Fing Limited
1st Floor Minerva House
Simmonscourt Road
Dublin 4, Ireland

email : sales@fing.com

www.fing.com

Offline DB

Fing Device Recognition Offline Database

FingDB
Last Update : 05 May 2021
Document Version : 2.1

Table of contents

1. Change History

Date	Version	Changes
3-Jul-2019	1.0	First Release.
28-Oct-2019	1.1	Added new fingerprint field “DHCPHOSTNAME” to dhcp_fingerprints and dhcp6_fingerprints tables.
06-Dec-2019	1.1	Change logo at the end
27-Dec-2019	1.2	New Device Type table section
14-Feb-2020	1.2	New Fing Ranking section
23-Mar-2020	1.3	New version 1.3 with new device recognition fields and improved recognition results
17-Sep-2020	1.4	New hostname_fingerprints table section
25-Nov-2020	1.4	Table of contents added, statistics updated, groups updated, table hostname_fingerprints described
25-Feb-2021	2.0	Obfuscation of Fing IP data Table indices optimization New metadata added for describing obfuscation algorithm
05-May-2021	2.1	Adding Python example for obfuscation

2. Introduction

Offline Database of Device Recognition is the offline version of Fing device recognition, designed to be used in any custom context: it is a snapshot of Fing device recognition fingerprints, stored in a SQLite® format.

The demo snapshot is limited to a small amount of the entire dataset for each recognition algorithm and should be used just for demo, validation and test purposes.

The Offline Database enables to design and implement custom recognition architecture by leveraging Fing fingerprint through the usage of standard SQL queries directly or by converting it into the format that suits consumer needs.

This document describes the Offline Database data model and provides sample queries for each recognition algorithm, in order to allow quick and frictionless evaluation and testing.

3. Fingerprints' description, size and limits

The Fing Offline Database contains several fingerprints related to some common network protocols:

- Clusters of MAC address for ARP
- Parameters List, Vendor and Hostname for DHCP v4 and v6
- User Agent header for HTTP
- Hostname for DNS

For further details on data gatherings please refer to standard RFCs or to Fing online documentation.

The fingerprints are organised in tables providing either the Fing identifier (type, brand, model, operating system) or the IEEE OUI Vendor, which is often used as filter in queries.

The Offline Database is approximately 30.5 GB in size.

The demo Offline Database, including a very small example portion of sample fingerprints, is just a few megabytes.

The table below provides counts of fingerprints for each fingerprint table.

Fingerprints table	Total fingerprints	Total demo fingerprints
dhcp_fingerprints	~1.3M	~70
dhcp6_fingerprints	~10.5K	~70
hostname_fingerprints	~590K	~70
hua_fingerprints	~59.5M	~30
mac_cluster	~22.2M	~1.9K
mac_vendors	~37.2K	~37.2K

The table below provides counts of device types and groups of devices.

Device table	Total devices ang groups	Total demo devices and groups
Different types	108	108
Different groups	8	8

4. Requirements

The Fing Offline Database is delivered as a SQLite file named fing_devrecog.db while the demo database is named fing_devrecog_demo.db.

Both can be queried either using a SQL client (e.g. DBrowser) or programmatically using the SQLite library in the proper language. For instance, in Java, you can use the JDBC connector provided from here: Xerial-SQLite-JDBC

Using a SQL Client

We use DBrowser: a high quality, visual, open-source tool to create, design, and edit database files compatible with SQLite.

- Download SQL Client from https://sqlitebrowser.org/dl/
- Connect to the Database: you just need to open the database and browse the file containing the database.
- Navigate through the database using the GUI. You just need to click on “Browse Data” to have a look at the data.

Using a Connector (example with Java)

It is needed to add in the class path the SQLite JDBC Connector which can be downloaded from here and use a snipper of code like the following to setup a connection.

```
import java.io.*;
import java.sql.*;

/**
* Setup a connection to the database
*
* @param dbFilePath the absolute path to the SQLite DB
* @return the Connection object
*/
public static Connection setupConnection(String dbFilePath) {
    Connection conn = null;
    File dbFile = new File(dbFilePath);
    if(!dbFile.exists()) {
        return null;
    }
    try {
        conn = DriverManager.getConnection("jdbc:sqlite:" + dbFilePath);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    return conn;
}
```

For further details we refer to Java SQL official documentation for the java.sql package.

5. Data Model

Device Recognition data is provided as a set of tables, one for each recognition algorithm.
In this paragraph we'll show, for each table, the list of columns with corresponding data type, the semantics and the primary purpose of the data in each column for, the recognition algorithm, etc.

Each table contains a set of columns to be queried to get the corresponding record(s) , a set of columns that specify recognition data (type, make, model, etc.) and a rank column to get the best results from the recognition.

The recognition rank is a measure from 0 to 100 of the quality of the recognition, but its scale is not a standard one like you could expect (e.g., where above 60 is sufficient and below 60 is not good): the higher and nearer to 100 is the better, but even lower values are acceptable. The recognition rank should always be used as a relative measure, among results in the same table or even in different tables, meaning that if r1 > r2, r1 is always better than r2.

For further information, please refer to Fing Ranking paragraph.

Column data types

The data in the Fing Offline database are of several types which we list hereafter:

- TEXT: string values of several lengths ranging from very short ones, up to longer portion.
- INTEGER: a whole number (positive) which typically acts as a classification feature of a table record, a logical Boolean flag or a rank value for recognition accuracy estimation.
- HASH: it's actually a TEXT type of data, hence a (quite long) string that is obtained (see below) by hashing a plain text with a suitable function and the ending the result using Base64 encoding scheme. As the Fing Offline Database contains short-lived Fing IP data that cannot be delivered as-they-are, the hashing trick, which is not reversible unless a long duration brute force approach is employed, guaranties the appropriate level of confidentiality and security. As we describe how we hash data when generating the Fing Offline Database, the same mechanism can be employed to hash "where condition" predicates in SQL queries so to search fingerprints against non-human understandable bits of information.

In the detailed description of the fingerprint tables in the following sections, we will report the type of data, according to the above classification, for each column in the described table.

How to generate hashed types

As mentioned above, some columns in the fingerprint's tables are irreversibly obfuscated by means of a hashing function transformation and a subsequent Base64 encoding to preserve security and confidentiality.

In order to execute the desired queries on tables having some column data obfuscated, every static part of the SQL query itself (e.g., where condition predicates) have to be obfuscated using the same process used when generating the entire database.
In the following Java and Python code snippets, we illustrate how the obfuscation can be carried out, provided that the field value of the column OBFUSCATION_SALT in the db_metadata table is used as a starting point of the obfuscation process. This piece of information will be the Base64 encoded salt used in the procedure below:

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

/**
 * Obfuscates a text data passed as an argument to this method
 * using a given Base64 salt, also provided as a parameter.
 *
 * @param plain the textual data to obfuscate.
 * @param salt the Base64 encoded salt to use.
 * @return the obfuscated text.
 */
private static String obfuscate(String plain, String salt) {
    // decode salt
    byte[] decodedSaltBytes = Base64.getDecoder().decode(salt);
    byte[] obfuscatedBytes;

    try {
        // obfuscate
        MessageDigest md = MessageDigest.getInstance("SHA-512");
        md.update(decodedSaltBytes);
        obfuscatedBytes = md.digest(plain.getBytes(StandardCharsets.UTF_8));
    } catch (NoSuchAlgorithmException var3) {
        // TODO: handle exception properly
        obfuscatedBytes = null;
    }

    // return base64 encoded and obfuscated string
    return new String(
        Base64.getEncoder().encode(obfuscatedBytes), StandardCharsets.UTF_8
    );
}
```

```python
import base64
import hashlib
import sys

def obfuscate(plain_text: str is not None, salt: str is not None) -> str:

    # get bytes from salt and decode using base 64
    decoded_salt_bytes = base64.b64decode(salt.encode('utf-8'))
```

```
    # get bytes from plain text
    plain_text_bytes = plain_text.encode('utf-8')

    # obfuscate plain text using salt and get obfuscated text bytes
    text_bytes = decoded_salt_bytes + plain_text_bytes
    obfuscated_text_bytes = hashlib.sha512(text_bytes).digest()

    # encode obfuscated text bytes using base 64 and get a string out of them
    obfuscated_text = base64.b64encode(obfuscated_text_bytes).decode('utf-8')
    return obfuscated_text
```

Tables

In this section we illustrate the details of the table data in the Fing Offline Database.

Table: db_metadata

This table stores utility info about the database instance in terms of global metadata.
This is useful to trace database creation date, version and SQLite version used to create it, in order to use the most suitable connector for such instance.
Moreover, the metadata about obfuscation recap the hashing and encoding schemes as well as the Base64 encoded salt to be used as just explained above.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples
CREATION_DATE	TEXT	The database creation date, in the format YYYY-MM-DD HH:MM:SS.SSS in UTC timezone.	2019-10-10 18:05:10:234
DB_VERSION	TEXT	The database version, used to track update info and changes.	1.0
SQLITE_VERSION	TEXT	The SQLite version used to create the database.	3.28.0
OBFUSCATION_ALGORITHM	TEXT	The name of the hashing function used to obfuscate.	**SHA-512**
OBFUSCATION_ENCODING	TEXT	The name of the encoding scheme used while obfuscating.	**Base64**
OBFUSCATION_SALT	TEXT	The Base64 encoded salt used while obfuscating. The salt MUST be decoded first before using to obfuscate data.	**ZmluZ19vZmZsaW5lX2RiMTc3ZDQwZGNkNjA=**

Table: device_type

This table contains the device type list used in the DEVICETYPE field of the other fingerprints' tables. It also includes for every device type, the caption (human-readable label) and the device type group. Groups are used to aggregate devices with similar features and usage patterns so that they can be retrieved together when querying the database and they are the following:

- Audio & Video
- Engineering
- Home & Office
- Industry
- Mobile
- Network
- Server
- Smart Home

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples
NAME	TEXT	The device type name, present in DEVICETYPE field of the fingerprint's tables.	SMART_PLUG
CAPTION	TEXT	The human-readable label for the device type.	Smart Plug
GROUP	TEXT	The belonging group.	Smart Home

Table: mac_vendor

You just need MAC address of a device in order to query this table: this is the simplest table, usually used in joining with the others, with only 2 columns used to get the mac vendor starting from the mac prefix, which in most cases is the first 3 bytes of a MAC Address but there is also a number of mac prefixes that are longer.
In the mac_vendor paragraph it will be shown how to deal with these different lengths.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples	Remarks
MACPREFIX	TEXT	Typically, the first 3 bytes of a MAC Address (but even longer).	307496	MAC prefix doesn't contain ":" character
VENDOR	TEXT	The vendor with such MACPREFIX.	Huawei	This is the MAC Vendor and not always corresponds to the brand

Table: mac_clusters

MAC clustering is the most powerful tool of Fing device recognition: it's the machine learning that allows Fing to increase the knowledge of IoT devices by leveraging their same fingerprints. It is based on predictive modelling, by correlation and supervised segmentation of MAC (hardware) address space.
mac_clusters table represents the fingerprints learnt by the mac clustering algorithm. Size of clusters is variable, and the example queries provided below allow you to always pick the best and sharpest available result.
The table also contains single devices seen and recognised by Fing with a 100% confidence. These are in this same table, with a cluster size of one. In case you need to reduce size of Offline DB, we strongly suggest to just cut out these single devices' fingerprints, as they are just identifying a single device each, as opposed to other mac clusters, where a single entry can qualify up to 16k different devices.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Type	Description	Examples	Remarks
MACCLUSTER	HASH	This field is composed of a MAC address with trailing 0 and a size. Size is used to get the number of possible MAC addresses in such cluster. Size 41 means that 48 - 41 = 7, so 2^7 possible MAC addresses that is 128.	Example (first 8 characters): BtTipc8Z Decoded text form: 6814017EA200/41	MAC prefix doesn't contain ":" character
MACPREFIX	HASH	The first 4 bytes of the MACCLUSTER.	Example (first 8 characters): 1Yq3a55p Decoded text form: 6814017E	MAC prefix doesn't contain ":" character
CLUSTERSIZE	INTEGER	How many possible MAC addresses are contained in this cluster. Smaller is this value, less different MAC addresses are contained in the cluster, this means that this cluster is more accurate in recognizing the right device.	128	CLUSTERSIZE equal to 1 means that the cluster is a MAC unique cluster, coming from SURE device
BITMASK	INTEGER	The bit mask used to match the right mac cluster, in	Bitmask most significant bits are 16 - (48-41) = 9:	MAC prefix is always 4 bytes (32 bits) and MAC

		decimal representation. The bit mask has 1 in the most significant bits according to the size in the MACCLUSTER field: if size is 41, less significant bits are 48 (the bits composing a MAC address) - 41 = 7. The bit mask will have 16 (the bits of the MAC suffix) - 7 = 9 most significant bits equal to 1 and the last 7 bits as the representation of the last 7 bits of the suffix in the MAC cluster.	111111111 Last 2 bytes are A200 and binary representation is: 1010001000000000 So, the resulting bitmask is: 1111111110000000 In decimal representation is 65408	suffix is always 2 bytes (16 bits)
BITVALUE	INTEGER	The MAC address' last 2 bytes decimal representation. This value must be matched when applying the bitmask to the MAC address to be recognized.	Last 2 bytes are A200, decimal representation is 41472	
DEVICETYPE	TEXT	The device type recognition.	TELEVISION	
DEVICEMAKE	TEXT	The device make recognition. Not always equals to the Vendor.	Sony	
DEVICEMODEL	TEXT	The device model recognition	KD-55XD8599	
DEVICEISFAMILY	INTEGER	1 if device is a family, e.g., iPhone, Galaxy 0 otherwise	1	
DEVICEOSNAME	TEXT	The device OS name recognition.	Android	
DEVICEOSVERS	TEXT	The device OS version recognition.	11.1.0	Always null in the current version
DEVICEOSBUILD	TEXT	The device OS build number recognition.	15B23	
DEVICERANK	INTEGER	The device rank for this record. It's computed from ranks	36	Under the same cluster size, better result is given by the greater rank

		coming from different recognition algorithms		

Table: dhcp_fingerprints table

This table represents the fingerprints related to the dhcp protocol.
In this table there are 2 different kind of vendors:

- those coming from the dhcp protocol itself
- those that is the real vendor which can be joined with the mac_vendor table to get a valid recognition result.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples	Remarks
DHCPPARAMS	HASH	A Comma Separated Value list of dhcp params sent by the device. The DHCP parameters list can be collected from the DHCP options of the DHCP request. The corresponding value is: **55**. A DHCP param is an octet (i.e., and integer smaller than 256). Please refers to DHCPv4 for any further detail.	Example (first 8 characters): R3rXxX7W Decoded text form: 53,55,57,61,50,51,12,255	
DHCPVENDOR	HASH	The DHCP Vendor class identifier can be collected from the DHCP options of the DHCP request. The corresponding value is: **43**. If the field is not present, you MUST not use it in the query.	Example (first 8 characters): hlK7ndk4 Decoded text form: android-dhcp-7.1.2	
DHCPHOSTNAME	HASH	The DHCP Hostname can be collected from the DHCP options of the DHCP request. The corresponding value is: **12.** If the field is not present, you MUST not use it in the query	Example (first 8 characters): M088XllG Decoded text form: Smashs-iPhone	

MACVENDOR	TEXT	The device MAC vendor. Use this in join with mac_vendor table, getting the vendor from the MAC address prefix to be recognized.	Apple	
DEVICETYPE	TEXT	The device type recognition.	MOBILE	
DEVICEMAKE	TEXT	The device make recognition. Not always equals to the Vendor.	Apple	
DEVICEMODEL	TEXT	The device model recognition.	iPhone 6	
DEVICEISFAMILY	INTEGER	1 if device is a family, e.g., iPhone, Galaxy 0 otherwise	1	
DEVICEOSNAME	TEXT	The device OS name recognition.	iOS	
DEVICEOSVERS	TEXT	The device OS version recognition.		Always null in the current version
DEVICEOSBUILD	TEXT	The device OS build number recognition.	15B23	
DEVICERANK	INTEGER	The device rank for this record.	30	Greater is the rank, greater is the result accuracy

Table: dhcp6_fingerprints

This table represents the fingerprints related to the dhcp6 protocol.
In this table there are 2 different kind of vendors:

- those coming from the dhcp6 protocol itself
- those that is the real vendor which can be joined with the mac_vendor table to get a valid recognition result.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples	Remarks
DHCPPARAMS	HASH	A concatenation of DHCPv6 options and DHCPv6 option requests with a '/' character.	Example (first 8 characters): J1WeF4Gj Decoded text form: 1:1,6,8,25/23,24,39	
DHCPVENDOR	HASH	It's a concatenation of the Enterprise ID with the Vendor Class Identifier from dhcp request in this order using a ':' character. If EID doesn't exist only the VCI is used.	Example (first 8 characters): THzeWZId Decoded text form: 311:MSFT 5.0	
DHCPHOSTNAME	HASH	The client hostname from dhcp.	Example (first 8 characters): Ac0cnIqQ Decoded text form: airport-home	
MACVENDOR	TEXT	The device MAC vendor. Use this in join with mac_vendor table, getting the vendor from the MAC address prefix to be recognized.	Apple	
DEVICETYPE	TEXT	The device type recognition.	WIFI_EXTENDER	
DEVICEMAKE	TEXT	The device make recognition. Not always equals to the Vendor.	Apple	

DEVICEMODEL	TEXT	The device model recognition.	AirPort	
DEVICEISFAMILY	INTEGER	1 if device is a family, e.g., iPhone, Galaxy 0 otherwise	1	
DEVICEOSNAME	TEXT	The device OS name recognition.	Android	
DEVICEOSVERS	TEXT	The device OS version recognition.	11.1.0	Always null in the current version
DEVICEOSBUILD	TEXT	The device OS build number recognition.	15B23	
DEVICERANK	INTEGER	The device rank for this record.	30	The greater the rank, the greater the result accuracy

Table: hua_fingerprints

This table represents the fingerprints related to the devices' http user agents.
The MACVENDOR field can be joined with the mac_vendor table to get a valid recognition result from device's MAC address.

Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples	Remarks
MACVENDOR	TEXT	The device MAC vendor. Use this in join with mac_vendor table, getting the vendor from the MAC address prefix to be recognized	Apple	
USERAGENT	HASH	The device's http user agent.	Example (first 8 characters): 8L2gBjB0 Decoded text form: itunesstored/1.0 iOS/11.4 model/iPhone7,1 hwp/t7000 build/15F79 (6; dt:107)	
DEVICETYPE	TEXT	The device type recognition.	MOBILE	
DEVICEMAKE	TEXT	The device make recognition. Not always equals to the Vendor.	Apple	
DEVICEMODEL	TEXT	The device model recognition.	iPhone 6 Plus	
DEVICEISFAMILY	INTEGER	1 if device is a family, e.g., iPhone, Galaxy 0 otherwise	1	
DEVICEMODELCODE	TEXT	The device model code from Android and Apple devices manufacturers		This field is deprecated
DEVICEOSNAME	TEXT	The device OS name recognition.	Android	

DEVICEOSVERS	TEXT	The device OS version recognition	11.1.0	
DEVICEOSBUILD	TEXT	The device OS build number recognition	15B23	
DEVICERANK	INTEGER	The device rank for this record	40	Greater is the rank, greater is the result accuracy

Table: hostname_fingerprints

This table represents the fingerprints related to the combination of MAC vendor and hostname (from DHCP or from DNS resolution). The queries must be performed by searching combinations of both fields mentioned above.
Hereafter, the fields in the table, each being defined and illustrated with an example.

Field name	Field Type	Description	Examples	Remarks
HOSTNAME	HASH	The client hostname from dhcp or DNS.	Example (first 8 characters): Ac0cnIqQ Decoded text form: airport-home	
MACVENDOR	TEXT	The device MAC vendor. Use this in join with mac_vendor table, getting the vendor from the MAC address prefix to be recognized	Apple	
DEVICETYPE	TEXT	The device type recognition.	WIFI_EXTENDER	
DEVICEMAKE	TEXT	The device make recognition. Not always equals to the Vendor.	Apple	
DEVICEMODEL	TEXT	The device model recognition.	AirPort	
DEVICEISFAMILY	INTEGER	1 if device is a family, e.g., iPhone, Galaxy 0 otherwise	1	
DEVICEOSNAME	TEXT	The device OS name recognition.	Android	
DEVICEOSVERS	TEXT	The device OS version recognition.	11.1.0	Always null in the current version.
DEVICEOSBUILD	TEXT	The device OS build number recognition.	15B23	
DEVICERANK	INTEGER	The device rank for this record.	30	The greater the rank, the greater the result accuracy.

6. Fing Ranking

Recognition ranking is a measure from 0 to 100 of the quality of the recognition. It is used to compare identifiers of the same device over time.

There are some peculiarities related to the Fing ranking that differs significantly from a standard linear evaluation system.

The Fing ranking has the following properties:

- The scale is not linear.
- **Any value greater than 0 is acceptable.**
- In case no recognition is available, no result is provided.
- **You can use comparison criteria such as "if r1 > r2", meaning r1 is the best solution.**

Some guidelines to understand the ranking:

- 90+ Highly accurate information, gathered directly from the devices or from processing precise information with machine learning.
- 40+ Very reliable protocol information (e.g., from SNMP, UPnP, Bonjour, HTTP User Agent fingerprints).
- 20+ DHCPv4/v6 information is based on best effort in most cases; however, in some other cases it is as strong as the protocols mentioned above.
- 1-20: NetBIOS, empiric rules (e.g., rules based on hostnames and MAC vendor).

Whilst processing the fingerprints and generating prediction models for recognition, our machine learning algorithms interpolate the above reference scores by weighting confidence levels.

It is advisable to always keep Fing results and not discard even low values given the drop is performed by the Fing engine itself in order to avoid sending a misrecognition.

The more protocols are provided, the more accurate recognition levels are achieved.

7. Query Examples

Device Recognition data is provided as a set of tables, one for each recognition algorithm.

In this paragraph we'll show for each table the list of columns with corresponding data type, what the data in each column is and for what it's best suited for, the recognition algorithm, etc.

Each table contains a set of columns to be queried to get the corresponding record(s), a rank to get the best results from the recognition and a set of columns that specify recognition data (type, make, model, etc.).

Table: mac_vendor

To query this table, since most of the mac prefixes have length of 3 bytes, but also exist mac prefixes that are longer, the following queries ensure that starting from the mac address the right result could be found.
Starting from 12 characters of the mac address (without ":"), keep in mind that prefixes in the mac_vendor table have length of 6, 7 and 9 characters.

Input:

1. MAC address: F0:23:B9:E3:10:5D
2. Remove colons: F023B9E3105D

Query:

```
select VENDOR
from mac_vendors
where 'F023B9E3105D' like MACPREFIX || '%'
order by LENGTH(MACPREFIX) desc
limit 1;
```

Result:

```
Domotz
```

Alternatively, for better performance, the following query can also be used, keeping in mind what said before about the mac prefix characters.
Take the substrings of the mac address looked up with lengths 6, 7 and 9.

Input:

1. MAC address: F0:23:B9:E3:10:5D
2. Remove colons: F023B9E3105D

Query:

```
select VENDOR
from mac_vendors
where MACPREFIX IN ('F023B9', 'F023B9E', 'F023B9E31')
order by LENGTH(MACPREFIX) desc
limit 1;
```

Result:

```
Domotz
```

Please note that in both query limit 1 is needed since more than one result is returned if the clause is not specified.

Table: mac_clusters

To get the correct mac clustering result, first get the BITVALUE, by using the '&' bitwise operator to apply the bitmask to the given MAC address suffix and then compare it with the BITVALUE for that cluster.
Then get the record with the smaller CLUSTERSIZE value.

Example 1

Input:

1. MAC address 801967D13F85
2. Extract Prefix: 801967D1
3. Extract Suffix (hex): 0x3F85
4. Obfuscate Prefix (just first 8 characters shown): oBMePNFV…

Query:

```
select CLUSTERSIZE,
DEVICETYPE,
DEVICEMAKE,
DEVICEMODEL,
DEVICEOSNAME,
DEVICERANK
from mac_clusters
where 'oBMePNFV…' = MACPREFIX and 0x3F85 & BITMASK = BITVALUE
order by CLUSTERSIZE;
```

Result:

CLUSTERSIZE	DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
4096	MOBILE	Infinix	Note 2	Android	25
16384	MOBILE	Infinix		Android	64

Example 2

Input:

1. MAC address 4040A74DD952
2. Extract Prefix: 4040A74D
3. Extract Suffix (hex): 0xD952
4. Obfuscate Prefix (just first 8 characters shown): CEzmPJbw…

Query:

```
select CLUSTERSIZE, DEVICETYPE, DEVICEMAKE, DEVICEMODEL,
DEVICEOSNAME, DEVICERANK
from mac_clusters
where 'CEzmPJbw…' = MACPREFIX and 0xD952 & BITMASK = BITVALUE
order by CLUSTERSIZE;
```

Result:

CLUSTERSIZE	DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
128	MOBILE	Sony		Android	38
512	MOBILE	Sony	Xperia M4 Aqua	Android	60
16384	MOBILE	Sony	Xperia	Android	45

Table: dhcp_fingerprints query

Example 1 (no hostname)

Input:

1. MAC address: 2C:9E:FC:BD:D1:A6
2. Remove colons: 2C9EFCBDD1A6
3. DHCP PARAMS: 1,3,6,28,15
4. Obfuscate DHCP PARAMS (just first 8 characters shown): amySovRQ…
5. DHCP VENDOR: Canon PRO-4100S
6. Obfuscate DHCP VENDOR (just first 8 characters shown): l6oPXqvq…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEOSNAME,
DEVICERANK
from dhcp_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('2C9EFCBDD1A6', 1, 6)
) and
DHCPPARAMS = 'amySovRQ…' and
DHCPVENDOR = 'l6oPXqvq…' and
DHCPHOSTNAME IS NULL
order by DEVICERANK desc;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
PRINTER	Canon	PRO-4100S		6

Example 2 (with hostname)

Input:

1. MAC address: D4:90:9C:BD:D1:A6
2. Remove colons: D4909CBDD1A6
3. DHCP PARAMS: 1,121,3,6,15,114,119,252
4. Obfuscate DHCP PARAMS (just first 8 characters shown): skJlIkkl…
5. DHCP HOSTNAME: Davids-iPad-2
6. Obfuscate DHCP HOSTNAME (just first 8 characters shown): sATuHIdy…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEOSNAME,
DEVICERANK
from dhcp_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('D4909CBDD1A6', 1, 6)
) and
DHCPPARAMS = '…' and
DHCPHOSTNAME = '…'
order by DEVICERANK desc;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
TABLET	Apple	iPad	iOS	4

Table: dhcp6_fingerprints query

Example 1 (no hostname)

Input:

1. MAC address: D0:53:49:96:DE:10
2. Remove colons: D0534996DE10
3. DHCP PARAMS: 1:8,1,3,16,6/17,23,24
4. Obfuscate DHCP PARAMS (just first 8 characters shown): /YOVAY0a…
5. DHCP VENDOR: 311:MSFT 5.0
6. Obfuscate DHCP VENDOR (just first 8 characters shown): THzeWZId…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEOSNAME,
DEVICERANK
from dhcp6_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('D0534996DE10', 1, 6)
) and DHCPPARAMS = '/YOVAY0a…' and
DHCPVENDOR = ' THzeWZId…' and
DHCPHOSTNAME IS NULL
order by DEVICERANK;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
COMPUTER			Windows	22

Example 2

Input:

1. MAC address: C8:63:F1:D3:F3:AE
2. Remove colons: C863F1D3F3AE
3. Message type: 1
4. Options list: 8,1,3,6
5. Options request: 24,23
6. Compute DHCP PARAMS: 1:8,1,3,6/24,23 (it's the concatenation of the three above attributes by : and /)
7. Obfuscate DHCP PARAMS (just first 8 characters shown): ix9AWhq7…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEOSNAME,
DEVICERANK
from dhcp6_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('C863F1D3F3AE', 1, 6)
) and DHCPPARAMS = 'ix9AWhq7…' and
DHCPVENDOR = ''
order by DEVICERANK;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
GAME_CONSOLE	Sony	Playstation 4		30

Example 3 (with hostname)

Input:

1. MAC address: 4C:0B:BE:B5:39:BE
2. Remove colons: 4C0BBEB539BE
3. DHCP PARAMS: 1:8,1,3,39,16,6'||'/'||'17,23,24,39
4. Obfuscate DHCP PARAMS (just first 8 characters shown): 9c47aoxv…
5. DHCP VENDOR: 311'||':'||'MSFT 5.0
6. Obfuscate DHCP VENDOR (just first 8 characters shown): THzeWZId…
7. DHCP HOSTNAME: XboxOne
8. Obfuscate DHCP HOSTNAME (just first 8 characters shown): oc1uomC5…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEOSNAME,
DEVICERANK
from dhcp6_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('4C0BBEB539BE', 1, 6)
) and
DHCPPARAMS = '9c47aoxv…' and
DHCPVENDOR = 'THzeWZId…' and
DHCPHOSTNAME = 'oc1uomC5…'
order by DEVICERANK;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
GAME_CONSOLE	Microsoft	Xbox One		5

Table: hua_fingerprints query

You can optionally use also the MAC vendor in the query on HTTP User-Agent table, but it's not mandatory.

Example 1

Input:

1. MAC address: B8:44:D9:24:00:78
2. Remove colons: B844D9240078
3. HTTP User Agent: itunesstored/1.0 iOS/11.4 model/iPhone7,1 hwp/t7000 build/15F79 (6; dt:107)
4. Obfuscate HTTP User Agent (just first 8 characters shown): 8L2gBjB0…

Query:

```
select DEVICETYPE, DEVICEMAKE, DEVICEMODEL, DEVICEMODELCODE,
DEVICEOSNAME, DEVICEOSVERS, DEVICERANK
from hua_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('B844D9240078', 1, 6)
) and
USERAGENT = '8L2gBjB0…'
order by DEVICERANK desc;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEMODELCODE	DEVICEOSNAME	DEVICEOSVERS	DEVICERANK
MOBILE	Apple	iPhone 6 Plus		iOS	11.3	40

Example 1 (without MACVERDOR table join)

Input:

1. MAC address: B8:44:D9:24:00:78
2. Remove colons: B844D9240078
3. HTTP User Agent: "Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1"
4. Obfuscate HTTP User Agent (just first 8 characters shown): SN02OegA…

Query:

```
select MACVENDOR, DEVICETYPE, DEVICEMAKE, DEVICEMODEL,
DEVICEOSNAME, DEVICEOSVERS, DEVICERANK
from hua_fingerprints
where USERAGENT = 'SN02OegA…'
order by DEVICERANK desc;
```

Result:

MACVENDOR	DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICEOSVERS	DEVICERANK
Apple	MOBILE	Apple	iPhone	iOS	11.0	20
Cisco	MOBILE	Apple	iPhone	iOS	11.0	20
StarTech.com	MOBILE	Apple	iPhone	iOS	11.0	20

Table: hostname_fingerprints query

Input:

1. MAC address: C8:D0:83:BD:D1:A6
2. Remove colons: C8D083BDD1A6
3. DHCP HOSTNAME: iphone-6s
4. Obfuscate DHCP HOSTNAME (just first 8 characters shown): 7ODiirS2…

Query:

```
select DEVICETYPE,
DEVICEMAKE,
DEVICEMODEL,
DEVICEOSNAME,
DEVICERANK
from hostname_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('C8D083BDD1A6', 1, 6)
)
and HOSTNAME = ' 7ODiirS2…'
order by DEVICERANK desc;
```

Result:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICERANK
MOBILE	Apple	iPhone 6S	iOS	10

Table: multiple

In order to get the best result from offline db, in case more than one protocol info is available, it's possible to run the following query on multiple tables. Sorting the result by rank in descending order returns the best result as first.
Blending data is still possible and gives the possibility to take different pieces of information from all the tables to get the best result possible.

Input:

1. MAC address: 30:74:96:BD:D1:A6
2. Remove colons: 307496BDD1A6
3. Extract Prefix: 307496BD
4. Extract Suffix (hex): 0xD1A6
5. Obfuscate Prefix (just first 8 characters shown): tdc5Zj3b…
6. DHCP PARAMS: '1,3,6,15,26,28,51,58,59,43'
7. Obfuscate DHCP PARAMS (just first 8 characters shown): aUJVrkqj…
8. DHCP VENDOR: 'HUAWEI:android:VTR'
9. Obfuscate DHCP VENDOR (just first 8 characters shown): xqp2dTIK…
10. HTTP User Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; VTR-L09 Build/HUAWEIVTR-L09)
11. Obfuscate HTTP User Agent (just first 8 characters shown): Z/L/rW+H…

Query:

```
select
'MACCLUSTER' as source, DEVICETYPE, DEVICEMAKE, DEVICEMODEL,
DEVICEOSNAME, DEVICEOSVERS, DEVICERANK
from mac_clusters
where ' tdc5Zj3b…' = MACPREFIX and
0xD1A6 & BITMASK = BITVALUE and
CLUSTERSIZE > 1
union
select
'DHCP' as source, DEVICETYPE, DEVICEMAKE, DEVICEMODEL,
DEVICEOSNAME, DEVICEOSVERS, DEVICERANK
from dhcp_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('307496BDD1A6', 1, 6)
) and
DHCPPARAMS = 'aUJVrkqj…' and
DHCPVENDOR = 'xqp2dTIK…'
union
select
'USER_AGENT' as source, DEVICETYPE, DEVICEMAKE, DEVICEMODEL,
DEVICEOSNAME, DEVICEOSVERS, DEVICERANK
```

```
from hua_fingerprints
where MACVENDOR = (
      select VENDOR
      from mac_vendors
      where MACPREFIX = SUBSTR('307496BDD1A6', 1, 6)
) and
USERAGENT = 'Z/L/rW+H…'
order by DEVICERANK desc;
```

Result:

SOURCE	DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICEOSVERS	DEVICERANK
USER_AGENT	MOBILE	Huawei	P10	Android	7.0 REL	39
MACCLUSTER	MOBILE	Huawei	P Series	Android		25
MACCLUSTER	MOBILE	Huawei	P10	Android		25
DHCP	MOBILE	Huawei	P10			19
DHCP	MOBILE	Huawei	P10	Android		5

As said earlier in this paragraph it's still possible to blend data to get the best possible result; in the above example the result would be:

DEVICETYPE	DEVICEMAKE	DEVICEMODEL	DEVICEOSNAME	DEVICEOSVERS
MOBILE	Huawei	P10	Android	7.0