Scalable Graph-Based Approach to Cross-Device Identification

Mr. Piyush Jinda[1]
*Director of Product Management InMobi Inc.*San Mateo, USA

***Abstract*—In the rapidly expanding digital environment, con- sumers frequently switch among multiple devicescomput- ers, tablets, and mobile phonesmaking it challenging for organizations to create a unified view of each user. This pa- per presents a framework for digital identification that combines deterministic rules, clickstream data, and big data technologies to stitch together user activities across diverse devices and platforms. Drawing on geo-location data, IP addresses, and frequency-of-visit patterns, the approach filters out unreliable connections (such as high- traffic or mobile IPs) and focuses on more stable Wi-Fi IP associations. These refined connections are processed using Hive for large-scale data storage and Spark GraphX for graph-based linkage analysis, with Oozie providing workflow automation. The resulting linkages receive confi- dence scores based on visit frequency and other contextual signals, enabling businesses to better personalize user experiences, improve ad targeting, and detect potential fraud. While the deterministic rules significantly enhance data quality, the paper also proposes future directions for more sophisticated techniquessuch as machine learningto incorporate behavioral and contextual features, further refining digital identification in large-scale production en- vironments.**

***Index Terms*—Digital Advertising, IP Identification, Location Identification, Cross Device Identification**

I. INTRODUCTION

With high presence of electronic devices like phone, tablets, computers, there is an increase in activity of digital data. As there are many devices owned by same users, data is scattered across many devices which makes it difficult of companies to make the complete profile of the users for digital marketing or ad-targeting purpose. There is a need to digitally identify those users so that a complete profile can be build which can be used to target those potential customers which were not known earlier[1]. The most usual way of identifying any user if the user will sign in to their website or applications so that companies can create a particular user id for that user so that they can track all the activities done by that user

but most of the time the user doesnt login until they have visited couple of times or they have particular interest in any product. Some companies also use the approach like email, phone or address provided in the profile to determine the guest but most of times, users leave those fields to be empty until they have any particular interest in ordering or there is any requirement for the profile to be completed before they can get what they need from the application or websites hence these solutions cant be used for different platforms or users.

Its really hard to determine a user if we dont have any user details to begin with. As technology grows, users mostly have different devices and every device can have multiple browsers. With users having the capability to clean their browsers cookies/device ids and resetting the browsers, it became hard to know if its the same user until they login from each and every device and browser, which is highly unlikely.

We need a different approach to understand the behav- ior depending on their visit patterns. We will be consider- ing the different approach which involve the information like IP address, device id, country, DMA and geo zip code, frequency count of the visit and difference. If we are able to identify one visit and relate all the remaining ones to the once which we identified earlier. This way we dont need the user to login from each and every device if user has logged in on once and has been sharing some of these attributes to make a common profile with all the devices, cookies tied to the once which we know about.

Identifying and building the user profile is very im- portant for many industries like retail, banking, trans- portation, health and social networking. They need those profiles so that they can personalize the user experiences as per their interest as well as users will get the benefit by getting the right recommendation from the companies without spending much time looking for the right product. The marketers can use

the user profile with 360- degree view and target the right ad to the right customers without spamming the users email id or sending irrelevant ads. They can have user clustered into right cluster depending on their activity which will be more effective than sending general campaign emails to all the users. In remaining chapters, we will be getting into the details of the analysis,
design of the solution.
Clickstream data is an information trail a user leaves behind while visiting a website and its typically captured in semi-structured website log files[2]. These website log files contain data elements such as a date stamp, Cookie ID or device id also known as mobile id, IP address, geo location and url etc. Clickstream data can be obtained by different ways, either using Adobe clickstream or google analytics. We will be using the clickstream data imported from Adobe directly into HDFS with extracted Cookie ID, Device ID, IP, Geo dma, Geo Zip as the initial data to be available for the identification to happen.

II. ANALYSIS

An HTTP Cookie (also known as a browser cookie) is a small piece of data sent from a website and stored in the user's web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful information or record a user's browsing activity [3]. Usually, the length of the cookie depends on the cookie ID sequence generator, which generally generates a lower cookie ID and an upper cookie ID (each length 19) concatenated by a colon.
A Device ID is a distinctive number associated with a smartphone or similar handheld device. Device IDs are different from hardware serial numbers (IMEIs). Each Ap- ple device has a unique device number known as a Unique Device Identifier (UDID), which is a 40-digit combination of letters and numbers. Android device IDs follow a similar pattern. Usually, these device ID numbers remain constant for the lifetime of the device unless a factory reset is performed.

A. Rule 1: Cookie ID vs Device ID Selection

Most companies have both websites and apps. Website data supplies cookie IDs, while app data supplies device IDs. Cookie IDs collected from websites are reliable, while cookies from an app environment are often not. Hence, for identification, prefer website cookie IDs and app device IDs, rather than using app-based cookie IDs (if present).

B. Rule 2: Removing Invalid Zip, DMA, IP, Cookie/De- vice ID

As clickstream captures logs, we often have null IP, geo_dma, geo_zip, or null cookie/device IDs. Because we rely on these columns for identification, any record lacking these is removed as invalid.

C. Rule 3: Removing Mobile Connections

Most devices are portable and may be accessed via cellular data while users are commuting. The IP address assigned during a cellular session is highly dynamic and can be reissued to different users within minutes. This makes mobile connections unreliable for user identifi- cation. Consequently, the process focuses on Wi-Fi IP addresses (whether static or dynamic). Static addresses remain the same until explicitly changed, whereas dynamic

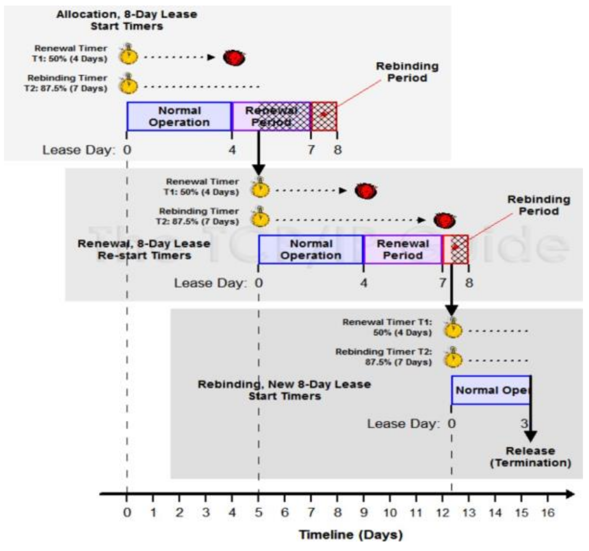
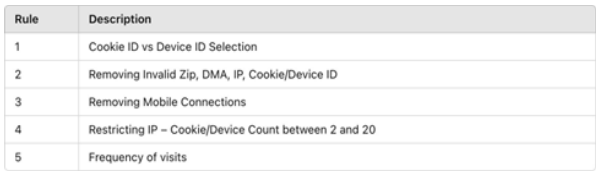


Fig. 1: Figure 1: DHCP Life Cycle



Rule	Description
1	Cookie ID vs Device ID Selection
2	Removing Invalid Zip, DMA, IP, Cookie/Device ID
3	Removing Mobile Connections
4	Restricting IP – Cookie/Device Count between 2 and 20
5	Frequency of visits

Fig. 2: Rule Table

addresses operate on a DHCP lease mechanism, which can still allow for some repeated usage patterns.

D. Rule 4: Restricting IP Cookie/Device Count Between 2 and 20

IP addresses alone are not entirely reliable for session tracking or user identification, because multiple machines can share the same public IP address via NAT (e.g., in offices, public Wi-Fi, etc.). To reduce ambiguous relation- ships, we only consider IP addresses that have between 2 and 20 cookies/devices mapped to them. Anything above 20 suggests a crowded IP (e.g., a coffee shop or large public hotspot) that could muddle user identification.

E. Rule 5: Frequency of Visits

Users sometimes access the same site from friends or family members networks. We use the frequency and timeframe of visits to further refine and filter out one-off or transient associations.

Below is a concise summary of the rules:

III. DESIGN

As a user visits a company website/app, clickstream data is logged. This data is stored in the Hadoop Dis- tributed File System (HDFS).

1) Initial ETL and Basic Authentication Data Perform basic cleansing and load cookie/device IDs that are definitively associated with logged-in users or other known attributes into a Hive table.
2) Extract Distinct IP From the above table, extract the distinct IP addresses visiting the site for the current day and store them in a Distinct IP table.
3) Generate Graph Input For each cookie ID, check that its cookie count per IP is between 2 and 20, that it is on Wi-Fi (i.e., not cellular), and that geo_dma, geo_zip, IP, and cookie_id are not null. Repeat the same for each device ID. Consolidate these valid cookie-IP and device-IP relationships into a Graph Input table. Collect all previously identified (logged- in) cookies/devices in a Cookie/Device-Handler ta- ble. Capture user visit frequency within a particular timeframe in a separate table.
4) Build Graph Using Spark GraphX Read the Graph Input table into Spark GraphX. Build a graph and gather neighboring or adjacent vertices to find possible cookie-cookie, cookie-device, or device- device connections. Store these relationships.
5) Filter/Incremental Extraction Compare newly discovered relationships with those from previous runs to find only incremental (new) associations.
6) Join with Known Identified Records Match new connections (from Step 5) with the Cookie/Device- Handler table (from Step 3.4) to tag previously
7) Scoring Combine results with frequency data (from Step 3.5) for confidence scoring. Store final matched pairs with assigned scores.
8) Automation Use Oozie or another scheduler to orchestrate the entire workflow, running daily (or at another chosen interval).

Below is an illustration of the design process:

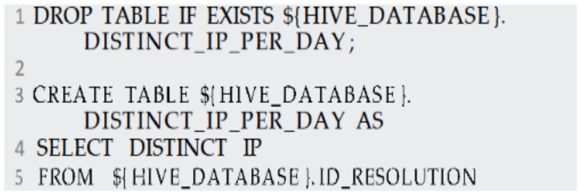
Table 2: Rule Implementation Table

IV. DEVELOPMENT

Development is currently in progress. Below are illus- trative steps and sample code used so far (written for Hadoop, Hive, Spark GraphX, and Oozie). *(Note: The code references are placeholders for demonstration and may require adjustments in your specific environment.)*

Step 1: Perform initial ETL extraction and load known cookies/devices with minimal cleansing. (Not shown in detail here.)

Step 2: Extract Distinct IP Addresses from Step 1 Table (for the current day).



```
DROP TABLE IF EXISTS ${HIVE_DATABASE}.
     DISTINCT_IP_PER_DAY;

CREATE TABLE ${HIVE_DATABASE}.
     DISTINCT_IP_PER_DAY AS
SELECT DISTINCT IP
FROM ${HIVE_DATABASE}.ID_RESOLUTION
```

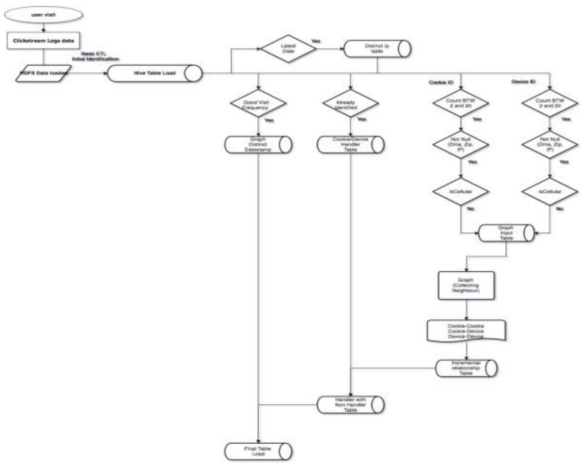
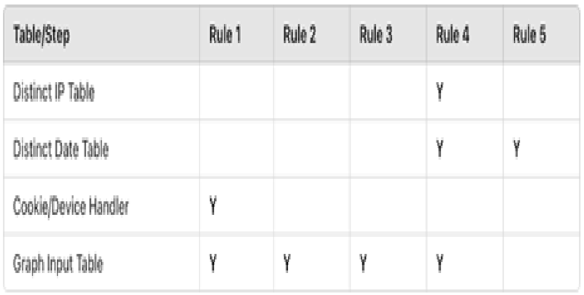


Fig. 3: Figure 2 -Design Steps



Table/Step	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Distinct IP Table				Y	
Distinct Date Table				Y	Y
Cookie/Device Handler	Y				
Graph Input Table	Y	Y	Y	Y	

Fig. 4: Rule Implementation Table

Step 3a/3c: For each cookie ID, ensure it meets us rules (220 count on IP, Wi-Fi, not null, etc.) and insert into a Graph Input table.

```
DROP TABLE IF EXISTS ${HIVE_DATABASE}.
      IP_GRAPH_INPUT_TABLE;

CREATE TABLE ${HIVE_DATABASE}.
      IP_GRAPH_INPUT_TABLE AS
SELECT DISTINCT
            trim(concat('IP',trim(HKIDTBL.
      geo_dma),'.',trim(HKIDTBL.geo_zip)
      ,'.',trim(HKIDTBL.ip))) AS value,
            trim(concat('CK',trim(HKIDTBL.
      cookie_id))) AS ID,
            HKIDTBL.stream_name
FROM ${HIVE_DATABASE}.ID_RESOLUTION
      HKIDTBL
WHERE HKIDTBL.geo_dma IS NOT NULL
   AND HKIDTBL.geo_zip IS NOT NULL
   AND HKIDTBL.ip IS NOT NULL
   AND HKIDTBL.cookie_id IS NOT NULL
   AND LENGTH(TRIM(HKIDTBL.cookie_id)) >
      38
   AND HKIDTBL.stream_name IS NOT NULL
   AND HKIDTBL.connection_type <> 4
   AND datestamp = date_sub(current_date
      ,2)
   AND HKIDTBL.ip IN
      (SELECT DISTINCT Ip_Count.ip
        FROM
          (SELECT geo_dma,
                  geo_zip,
                  INTERIM.ip,
                  count(distinct(cookie_id))
      AS cnt
            FROM ${HIVE_DATABASE}.
      ID_RESOLUTION INTERIM
            WHERE INTERIM.geo_dma IS NOT NULL
              AND INTERIM.geo_zip IS NOT NULL
              AND INTERIM.ip IS NOT NULL
              AND INTERIM.cookie_id IS NOT
      NULL
              AND INTERIM.stream_name IS NOT
      NULL
              AND INTERIM.connection_type <>
      4
              AND datestamp = date_sub(
      current_date,2)
            GROUP BY INTERIM.ip,
                     geo_zip,
                     geo_dma
            HAVING cnt BETWEEN 2 AND 20
          ) Ip_Count
      );
```

Step 3b/3c: Same logic for device IDs:

```
INSERT INTO TABLE ${HIVE_DATABASE}.
      IP_GRAPH_INPUT_TABLE
SELECT DISTINCT
            trim(concat('IP',trim(HKIDTBL.
      geo_dma),'.',trim(HKIDTBL.geo_zip)
      ,'.',trim(HKIDTBL.ip))) AS value,
            trim(concat('DV',trim(HKIDTBL.
      mobile_uuid))) AS ID,
            HKIDTBL.stream_name
FROM ${HIVE_DATABASE}.ID_RESOLUTION
      HKIDTBL
WHERE HKIDTBL.geo_dma IS NOT NULL
   AND HKIDTBL.geo_zip IS NOT NULL
   AND HKIDTBL.ip IS NOT NULL
   AND HKIDTBL.mobile_uuid !=
      '00000000-0000-0000-0000-000000000000'
   AND HKIDTBL.mobile_uuid IS NOT NULL
   AND LENGTH(TRIM(HKIDTBL.cookie_id)) <
      38
   AND HKIDTBL.stream_name IS NOT NULL
   AND HKIDTBL.connection_type <> 4
   AND datestamp = date_sub(current_date
      ,2)
   AND HKIDTBL.ip IN
      (SELECT DISTINCT Ip_Count.ip
        FROM
          (SELECT geo_dma,
                  geo_zip,
                  INTERIM.ip,
                  count(distinct(mobile_uuid
      )) AS cnt
            FROM ${HIVE_DATABASE}.
      ID_RESOLUTION INTERIM
            WHERE INTERIM.geo_dma IS NOT NULL
              AND INTERIM.geo_zip IS NOT NULL
              AND INTERIM.ip IS NOT NULL
              AND INTERIM.mobile_uuid IS NOT
      NULL
              AND INTERIM.stream_name IS NOT
      NULL
```

```
            AND INTERIM.connection_type <>
    4
            AND datestamp = date_sub(
    current_date ,2)
         GROUP BY INTERIM.ip,
                  geo_zip,
                  geo_dma
         HAVING cnt BETWEEN 2 AND 20
       ) Ip_Count
   );
```

Step 3d: Identify cookies/devices that have already been definitively linked to a known user (logged in, known email, etc.):

```
DROP TABLE IF EXISTS ${HIVE_DATABASE}.
    ID_RESOLUTION_IP_v2;

CREATE TABLE ${HIVE_DATABASE}.
    ID_RESOLUTION_IP_v2 AS
SELECT *
FROM
(
    SELECT *,
            row_number() OVER (PARTITION BY
    id, ip, geo_dma, geo_zip ORDER BY
    datestamp) AS rownum
    FROM
    (
        SELECT *,
               CASE
                    WHEN mobile_uuid IS NOT
    NULL
                         AND (LENGTH(TRIM(
    cookie_id)) < 38 OR TRIM(cookie_id) IS
     NULL)
                    THEN mobile_uuid
                    WHEN LENGTH(TRIM(
    cookie_id)) >= 38
                    THEN COOKIE_ID
                    ELSE NULL
               END AS ID
        FROM ${HIVE_DATABASE}.ID_RESOLUTION
    ) A
    WHERE ((CUST_GST_ID IS NOT NULL AND
    CUST_GST_ID['1'] IS NOT NULL )
            OR (EMAIL_GST_ID IS NOT NULL
    AND EMAIL_GST_ID['1'] IS NOT NULL )
            OR (ORDER_GST_ID IS NOT NULL
    AND ORDER_GST_ID['1'] IS NOT NULL )
            OR (REGISTRY_GST_ID IS NOT NULL
     AND REGISTRY_GST_ID['1'] IS NOT NULL
    )
            OR (CW_GST_ID IS NOT NULL AND
    CW_GST_ID ['1'] IS NOT NULL ))
      AND datestamp BETWEEN date_sub(
    current_date,182) AND date_sub(
    current_date ,2)
      AND A.ip IN (SELECT Ip FROM ${
    HIVE_DATABASE}.DISTINCT_IP_PER_DAY)
) A
WHERE rownum = 1;
```

Step 3e: Compute visit frequency and confidence:

```
DROP TABLE IF EXISTS ${HIVE_DATABASE}.
    graph_distinct_datestamp;

CREATE TABLE ${HIVE_DATABASE}.
    graph_distinct_datestamp AS
SELECT A.*,
        CASE
            WHEN DateDiffer >= 7 AND CNT >
    2 THEN 0.95
            WHEN DateDiffer >= 4 AND
    DateDiffer <= 6 AND CNT > 2 THEN 0.85
            WHEN DateDiffer < 4 AND CNT >
    2 THEN 0.5
            WHEN DateDiffer >= 7 AND CNT =
    2 THEN 0.5
            WHEN DateDiffer >= 4 AND
    DateDiffer <= 6 AND CNT = 2 THEN 0.35
            WHEN DateDiffer < 4 AND CNT =
    2 THEN 0.25
            WHEN DateDiffer = 1 AND CNT =
    1 THEN 0.1
            WHEN DateDiffer > 1 AND CNT =
    1 THEN 0.2
            ELSE 0.01
        END AS CONFIDENCE
FROM
(
    SELECT UPPER(ID) AS ID,
           IP,
           geo_dma ,
           geo_zip,
           DATEDIFF(MAX(DATESTAMP), MIN(
    DATESTAMP)) AS DateDiffer,
           COUNT(DISTINCT(DATESTAMP)) AS
    CNT
    FROM
    (
        SELECT *,
               CASE
                   WHEN mobile_uuid IS NOT
    NULL
                        AND (LENGTH(TRIM(
    cookie_id)) < 38 OR TRIM(cookie_id) IS
     NULL)
                   THEN mobile_uuid
                   WHEN LENGTH(TRIM(
    cookie_id)) >= 38
                   THEN COOKIE_ID
                   ELSE NULL
               END AS ID
        FROM ${HIVE_DATABASE}.id_resolution
        WHERE datestamp BETWEEN date_sub(
    current_date,182) AND date_sub(
    current_date ,2)
    ) A
    WHERE ip IN ( SELECT ip FROM ${
    HIVE_DATABASE}.DISTINCT_IP_PER_DAY)
     AND datestamp BETWEEN date_sub (
    current_date ,182) AND date_sub (
    current_date ,2)
     AND (cookie_id IS NOT NULL OR
    mobile_uuid IS NOT NULL)
     AND ip IS NOT NULL
     AND geo_dma IS NOT NULL
     AND geo_zip IS NOT NULL
     AND connection_type <> 4
    GROUP BY ID, ip, geo_zip, geo_dma
) A;
```

Step 4: Use Spark GraphX to create a graph from the Graph Input table, then collect neighbors (cookiecookie, cookiedevice, devicedevice). Below is example Scala/Spark code:

```
import java.nio.charset.Charset
import org.apache.spark.graphx.PartitionStrategy.RandomVertexCut
import org.apache.spark.graphx._
import org.apache.spark.sql.hive.HiveContext
import org.apache.spark.{SparkContext, SparkConf}

object HashkeyGraphXG2C {

  def main(args: Array[String]) {
    val conf = new SparkConf()
    val sc = new SparkContext(conf)
    val sqlContext = new HiveContext(sc)
    import sqlContext.implicits._

    def hashId(str: Any) = {
      com.google.common.hash.Hashing.md5.hashString(str.toString, Charset.defaultCharset()).asLong()
    }

    // Create Vertex RDD
    val df_vertex_source = sqlContext.sql("""
      SELECT value FROM ${HIVE_DATABASE}.IP_GRAPH_INPUT_TABLE
      UNION ALL
      SELECT ID AS value FROM ${HIVE_DATABASE}.IP_GRAPH_INPUT_TABLE
    """).distinct()

    val vertices = df_vertex_source.map(row => (hashId(row(0)), row(0).toString)).cache()

    // Create Edge RDD
    val df_edge_source = sqlContext.sql("""
     SELECT value, ID, stream_name FROM ${HIVE_DATABASE}.IP_GRAPH_INPUT_TABLE
    """).distinct()

    val edges = df_edge_source.map(row => Edge(hashId(row(0)), hashId(row(1)), row(2).toString))

    val inputGraph = Graph(vertices, edges).partitionBy(RandomVertexCut).groupEdges((a, b) => a + b)

    // Collect neighbor IDs
    val verticesWithSuccessors: VertexRDD[Array[VertexId]] =
      inputGraph.collectNeighborIds(EdgeDirection.Out)

    val successorSetGraph = Graph(verticesWithSuccessors, edges)

    // mapReduceTriplets to find second-degree neighbors, for example
    val ngVertices: VertexRDD[Set[VertexId]] =
      successorSetGraph.mapReduceTriplets[Set[VertexId]](
        triplet => Iterator((triplet.dstId, triplet.srcAttr.toSet)),
        (s1, s2) => s1 ++ s2
      ).mapValues((id, neighbors) => neighbors - id)

    val ngEdges = ngVertices.flatMap {
      case (source: VertexId, allDests: Set[VertexId]) =>
        allDests.map(dest => Edge(source, dest, ""))
    }

    // Example: Saving results for further processing
    ngEdges.map(e => (e.srcId, e.dstId))
      .join(vertices)
      .map { case (srcId, (dstId, srcVal)) => (dstId, srcVal) }
      .join(vertices)
      .map { case (dstId, (srcVal, dstVal)) => (srcVal, dstVal) }
      .saveAsTextFile("/user/SVHKYANP/Ip2Google")

    sc.stop()
  }
}
```

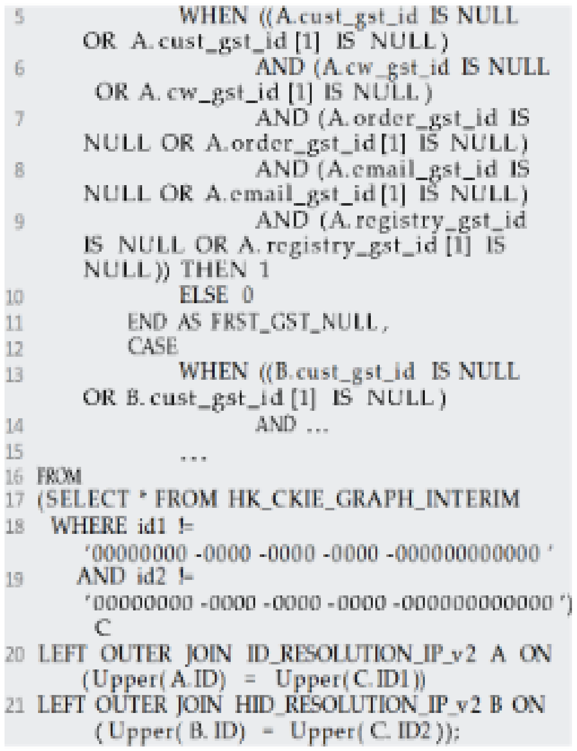
Step 5: Identify incremental relationships by comparing newly detected pairs to what you already have.

```
DROP TABLE HK_CKIE_GRAPH_INTERIM;

CREATE TABLE HK_CKIE_GRAPH_INTERIM AS
SELECT ID1, ID2
FROM (
    SELECT A.id1 AS ID1,
           A.id2 AS ID2,
           B.id1 AS B_ID1,
           B.id2 AS B_ID2
    FROM (
        SELECT UPPER(SUBSTR(ck2,4,length(ck2)-5)) AS id2,
               UPPER(SUBSTR(ck1,5,length(ck1)-5)) AS id1
        FROM CKIE_IP_graph
    ) A
    LEFT OUTER JOIN graph_IP_Resolution B
    ON A.ID1 = B.ID1 AND A.ID2 = B.ID2
) X
WHERE B_ID1 IS NULL;
```

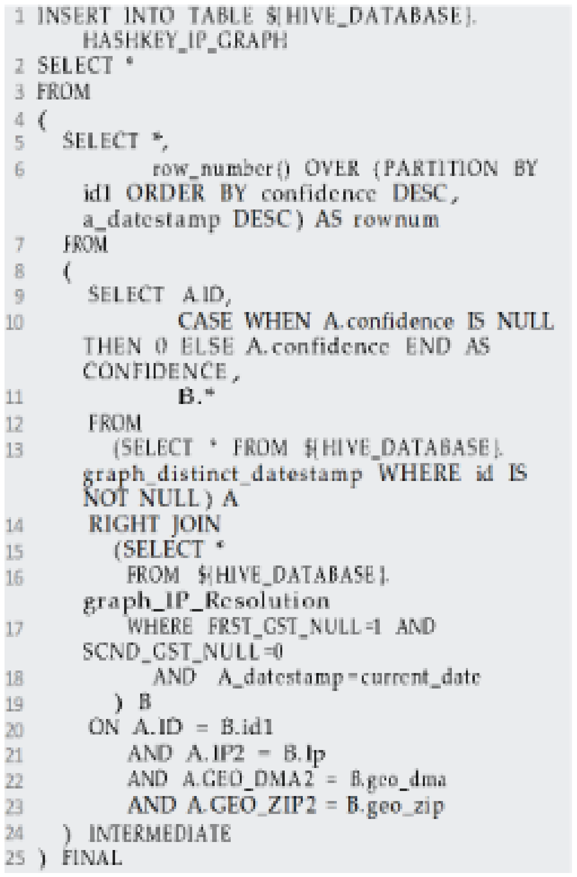
Step 6: Join incremental relationships to the already identified records, so that any newly discovered cookie/device can be tied to a known user ID.

```
INSERT INTO TABLE ${HIVE_DATABASE}.graph_IP_Resolution
SELECT Upper(C.ID1) AS ID1,
       Upper(C.ID2) AS ID2,
       CASE
```

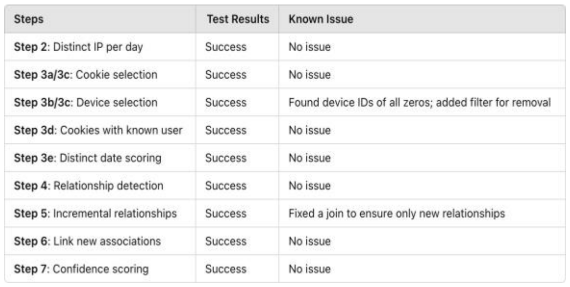


```
            WHEN ((A.cust_gst_id IS NULL
     OR A.cust_gst_id [1] IS NULL)
                  AND (A.cw_gst_id IS NULL
      OR A.cw_gst_id [1] IS NULL)
                  AND (A.order_gst_id IS
     NULL OR A.order_gst_id[1] IS NULL)
                  AND (A.email_gst_id IS
     NULL OR A.email_gst_id[1] IS NULL)
                  AND (A.registry_gst_id
     IS NULL OR A.registry_gst_id [1] IS
     NULL)) THEN 1
            ELSE 0
        END AS FRST_GST_NULL,
        CASE
            WHEN ((B.cust_gst_id IS NULL
     OR B.cust_gst_id [1] IS NULL)
                  AND ...
            ...
FROM
(SELECT * FROM HK_CKIE_GRAPH_INTERIM
  WHERE id1 !=
     '00000000 -0000 -0000 -0000 -000000000000 '
    AND id2 !=
     '00000000 -0000 -0000 -0000 -000000000000 ')
      C
LEFT OUTER JOIN ID_RESOLUTION_IP_v2 A ON
     (Upper(A.ID) = Upper(C.ID1))
LEFT OUTER JOIN HID_RESOLUTION_IP_v2 B ON
      (Upper(B.ID) = Upper(C.ID2));
```

Step 7: Combine with frequency data to apply confidence scores:

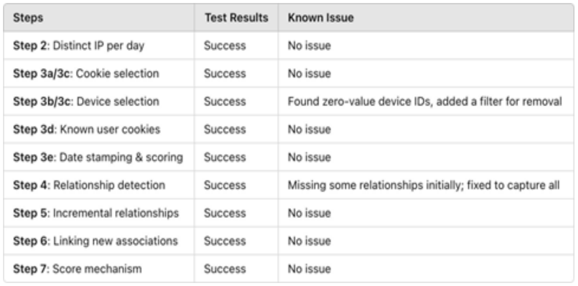


```
INSERT INTO TABLE ${HIVE_DATABASE}.
     HASHKEY_IP_GRAPH
SELECT *
FROM
(
   SELECT *,
           row_number() OVER (PARTITION BY
     id1 ORDER BY confidence DESC,
     a_datestamp DESC) AS rownum
   FROM
   (
    SELECT A.ID,
            CASE WHEN A.confidence IS NULL
     THEN 0 ELSE A.confidence END AS
     CONFIDENCE,
            B.*
    FROM
      (SELECT * FROM ${HIVE_DATABASE}.
     graph_distinct_datestamp WHERE id IS
     NOT NULL) A
    RIGHT JOIN
      (SELECT *
       FROM ${HIVE_DATABASE}.
     graph_IP_Resolution
       WHERE FRST_GST_NULL=1 AND
     SCND_GST_NULL=0
         AND A_datestamp=current_date
      ) B
    ON A.ID = B.id1
       AND A.IP2 = B.Ip
       AND A.GEO_DMA2 = B.geo_dma
       AND A.GEO_ZIP2 = B.geo_zip
   ) INTERMEDIATE
) FINAL
```



Steps	Test Results	Known Issue
Step 2: Distinct IP per day	Success	No issue
Step 3a/3c: Cookie selection	Success	No issue
Step 3b/3c: Device selection	Success	Found device IDs of all zeros; added filter for removal
Step 3d: Cookies with known user	Success	No issue
Step 3e: Distinct date scoring	Success	No issue
Step 4: Relationship detection	Success	No issue
Step 5: Incremental relationships	Success	Fixed a join to ensure only new relationships
Step 6: Link new associations	Success	No issue
Step 7: Confidence scoring	Success	No issue

Fig. 5



Steps	Test Results	Known Issue
Step 2: Distinct IP per day	Success	No issue
Step 3a/3c: Cookie selection	Success	No issue
Step 3b/3c: Device selection	Success	Found zero-value device IDs, added a filter for removal
Step 3d: Known user cookies	Success	No issue
Step 3e: Date stamping & scoring	Success	No issue
Step 4: Relationship detection	Success	Missing some relationships initially; fixed to capture all
Step 5: Incremental relationships	Success	No issue
Step 6: Linking new associations	Success	No issue
Step 7: Score mechanism	Success	No issue

Fig. 6

Step 8: Automate the entire workflow with Oozie (including a Coordinator definition for scheduling). *(Full Oozie XML omitted here for brevity, but see the PDF for a complete example.)*

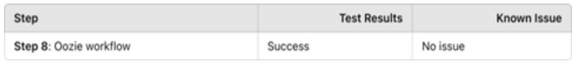
V. TESTING

Unit Testing Data Validation
Automation Testing

VI. SUMMARY

Digital Identification helps multiple industries (re- tail, banking, transportation, etc.) customize user expe- riences and detect potential fraud by creating a 360- degree view of each user across all devices. Traditionally, marketing or fraud detection efforts rely on user-supplied data, which may be limited. As technology grows and devices proliferate, a tool such as this becomes essential to unify disparate data points.

Rules and filtering steps (cookie validity, IP constraints, device usage, etc.) help ensure accuracy, removing untrust- worthy associations. Although this approach may lower total coverage (because overly filtered connections get dropped), the quality of the matches is higher. Once a user



Step	Test Results	Known Issue
Step 8: Oozie workflow	Success	No issue

Fig. 7

is identified on one device or cookie, the system automat- ically links the users additional devices (when used under similar Wi-Fi/IP constraints) without requiring multiple logins.
We leverage Big Data technologies (Hadoop, Hive, Spark GraphX, Oozie) for efficient, large-scale processing. This approach scales to handle trillions of records with relatively modest hardware costs.

Conclusion: This tool is straightforward to operate, requiring minimal data (geo location plus device/cookie fields). By automating daily, it continually refines cross- device relationships to help your organization benefit from a richer understanding of each user.

VII. CONCLUSIONS AND RECOMMENDATIONS

In this tool, we have presented the analysis, design, and partial implementation of a system to match devices and cookies to digitally identify userssolving the multi-device tracking problem for many businesses.

Recommendations:

1) Use the filter rules in Table 1 as a baseline prior to finalizing or updating customer profiles.
2) If data volume is high, deploy Big Data technologies (Hive, Spark, etc.) for scalability.
3) Consider that not all identified relationships are 100% certain. Some unknown fraction of false pos- itives may arise from shared IP addresses. Use or weight relationships by confidence scores.
4) Expand or alter these rules to suit your particular user environment.

DIRECTIONS OF FUTURE WORK

1) Enhanced Feature Engineering Incorporate user behavior patterns and interests (beyond IP and geo data) for more robust clustering of devices to users.
2) Machine Learning Models Employ Random For- est or ensemble methods for probabilistic matching, potentially capturing more valid connections than deterministic rules.
3) Mobile Connection Logic Consider partial us- age of cellular IP addresses, with lower confidence weighting, to capture on-the-go usage patterns.
4) Crowded IP Considerations Instead of discard- ing high-traffic IP addresses outright, incorporate advanced heuristics or ML to detect repeated ses- sions from the same device.
5) Performance Optimizations Explore partition- ing, indexing, or advanced caching strategies (e.g., RDD caching in Spark) to speed large-scale process- ing.

REFERENCES

[1] Mohammad Kamrul Islam, Aravind Srinivasan. Apache Oozie: The Workflow Scheduler for Hadoop. OReilly Me- dia, 2015.

[2] John S. McKean. Customer’s New Voice: Extreme Rel- evancy and Experience through Volunteered Customer Information. Wiley, 2015.

[3] Jeff Larson, Stuart Draper. Internet Marketing Es- sentials: A Comprehensive Digital Marketing Textbook. Stukent, Inc., 2015.

[4] Asim Ansari, Carl F. Mela. E-Customization. *Jour- nal of Marketing Research*, May 2003, Vol. 40, No. 2.

[5] Cross-Device Tracking: Matching Devices and Cookies https://arxiv.org/pdf/1510.01175.pdf

[6] VISUALIZE WEBSITE CLICKSTREAM DATA

[7] http://hortonworks.com/hadoop-tutorial/how-to-visualize-website-clickstream-data/

[8] DHCP Lease Life Cycle Overview http://www.tcpipguide.com/free/t_DHCPLeaseLifeCycle- OverviewAllocationReallocationRe.htm