North American Journal of Engineering and Research

Est. 2020

A Novel Approach for Efficient Conversion Tracking in Mobile Applications: Leveraging Event Linking and Beacon Events

Sambu Patach Arrojula

Email: sambunikhila@gmail.com

Abstract

In the context of analytics, determining the source of a conversion, such as a user registration or product purchase, is a critical yet challenging task. Applications typically have multiple entry points leading to these important actions, such as various ads or UI elements. Product owners seek to identify which entry points are most effective in driving conversions. Traditionally, applications track these entry points through complex event instrumentation that propagates data from screen to screen, resulting in high maintenance overhead and increased potential for errors. In this paper, we propose a novel design that simplifies event instrumentation while efficiently determining the source entry points for conversions. We demonstrate how this design can be implemented on both the client side and the backend, ensuring robust conversion tracking through analytic events.

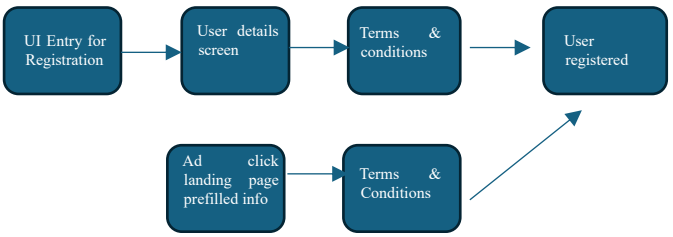
Keywords: Mobile application, analytic events, Conversion tracking, Event linking, User journey, Analytic cache.

Introduction

Conversion tracking is a key component of analytics in mobile applications, particularly for critical user actions such as registrations and purchases. Application owners would like to track and know about the entry points that are driving most of these important conversions. These applications often have multiple pathways leading to the same conversion point, making it challenging to accurately track the source of a conversion. For instance, in scenarios where a conversion can occur through different sequences of screens e.g., A→B→C→M and X→Y→Z→M where M is a conversion event, it becomes essential to identify whether the conversion originated from path A or X.

Traditional approaches involve embedding detailed tracking information in each UI element, which must be propagated throughout the application. This method is not only complex and prone to errors but also rigid, as it relies on pre-compiled data that cannot be easily modified or extended. For example, when paths A→B→C→D and E→B→C→D share common elements B and C, the complexity of the instrumentation B & C increases significantly, leading to maintenance challenges and a higher likelihood of bugs.

This paper presents a new approach that leverages client-side caching and event linking to efficiently trace the origin of conversions. By maintaining a logical sequence of events in the client's analytics cache, we can backtrack through user actions to determine the source of any conversion event. This approach not only simplifies the instrumentation process but also enhances flexibility, allowing for the dynamic retrieval of additional information even after the event has occurred.



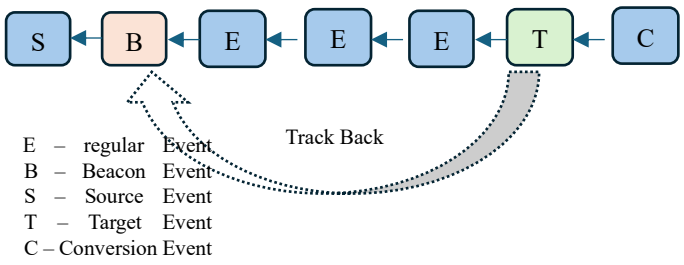
Example of a user journey for registration with two entry points

Approch

In this solution we assume the presence of an analytics framework on the client side that incorporates a caching mechanism. This caching system allows for events to be temporarily stored and processed locally before being uploaded to the backend in batches, rather than being transmitted immediately as they occur. This approach is beneficial in several ways, including reducing network usage, improving performance, and allowing for more sophisticated processing of events before they are sent to the server. By leveraging this caching component, we can implement more complex event handling strategies that would not be feasible in a real-time upload scenario.

Our approach centers on the concept of linking analytics events in a chronological sequence within the client's cache. When an event is fired, it is not treated as an isolated occurrence; instead, it is linked to the event that immediately preceded it. This chronological linking creates a logical chain of events that reflects the user's journey through the application. By maintaining this chain, the system can backtrack through the sequence of events to identify specific points of interest, such as the origin of a conversion event. This linking mechanism is key to our solution, as it eliminates the need for complex and error-prone instrumentation across multiple screens or steps within the application.

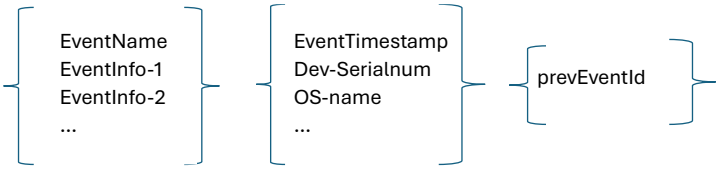
In addition to chronological linking, our approach relies on the application to instrument beacon events at critical checkpoints within the user's journey. These beacon events serve as markers or reference points that denote potential conversion paths. When a conversion event occurs, the system can trace back through the linked events to locate the corresponding beacon event, thereby identifying the source of the conversion. This method greatly simplifies the instrumentation process, as intermediate screens or steps do not require any special handling or the propagation of information from one event to another. By focusing on beacon events and chronological linking, we provide a streamlined and efficient solution for tracking the origins of important conversion events without the overhead of maintaining detailed instrumentation across the entire application.



Details

Event Structure

In our proposed solution, each event captured by the application consists of two main components: application-specific data and analytics framework metadata. The application-specific data is designed to be flexible, enabling the app to fill in key-value pairs that reflect the specific details of a user interaction, such as the action performed or the screen viewed. This part of the event structure is customizable, allowing different applications to tailor the event data according to their unique requirements. On the other hand, the analytics framework metadata is standardized, providing essential contextual information like the timestamp of the event, device characteristics, operating system details, and other environment-specific data. This metadata remains consistent across events, offering a reliable basis for analyzing user behavior and interactions. To facilitate the backtracking mechanism central to our solution, the analytics framework introduces an additional field called prevEvent. This field is automatically populated by the framework before the event is added to the cache and contains the identifier of the immediately preceding event. By including this prevEvent entry, the framework effectively links each event to its predecessor, creating a chronological sequence of events. This linkage allows the system to easily navigate through the user's interaction history within the application.

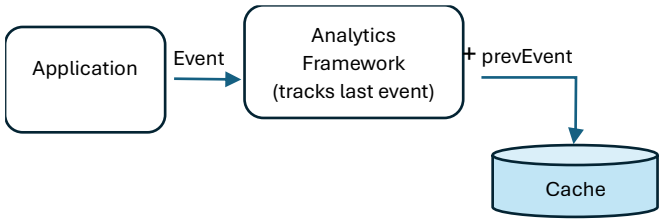


Event linking

The analytics framework is responsible for maintaining a continuous link between events by consistently tracking the "previous event" as users interact with the application. Each time a new event is generated, the framework automatically populates it with the identifier of the existing previous event, effectively linking the current event to the one that occurred immediately before it. Once this linkage is established, the framework updates its internal record, replacing the previous event with the current one. This process ensures that a chronological chain of events is maintained, allowing for seamless backtracking through the user's interaction history.

When the analytics framework is integrated with multiple applications on the same client, it manages separate "previous event" records for each registered application. This means that events generated by different applications are linked to their respective preceding events within that specific application's context. By maintaining these distinct chains, the framework ensures that each application's event history remains accurate and independent, allowing for precise tracking and analysis.

However, in real-world scenarios, there is always a possibility of bugs or errors on the client side that could disrupt this event linking process. If such an issue arises and an event becomes delinked—meaning the expected preceding event is missing—the framework has a mitigation strategy in place. In such cases, the backend system is equipped to search for the next available preceding event from the same client and application. This search ensures that even if a link is broken, the overall chain can be restored, minimizing the impact on the accuracy and reliability of conversion tracking. This approach provides a robust mechanism for handling potential errors, ensuring that the event linking process remains resilient and effective even in the face of unexpected client-side issues.



Beacon Event

In the proposed analytics framework, a new event type, BEACON_EVENT, is introduced to enhance the tracking of conversion paths. This event is specifically designed to be emitted by the application whenever it detects that a user has entered a potential conversion path. The purpose of the BEACON_EVENT is to serve as a marker that can be referenced later when the conversion is completed, allowing the system to trace the conversion back to its starting point using the event-linking mechanism.

Applications are expected to instrument BEACON_EVENTs at the entry points of all known conversion paths. For example, when a user initiates the first action in a conversion path, such as starting a sign-up process, the application first emits the regular event corresponding to this action—say, a SIGN_UP_START event. Immediately following this, the application emits a BEACON_EVENT that includes the identifier of the preceding SIGN_UP_START event as well as the name of the expected end event-name for that conversion path, such as REGISTRATION_SUCCESS.

```
EventName: BEACON_EVENT
SourceEventID: <eventid>
TargetEventName: <eventName>
```

Conversion Event and tracing back

The analytics framework introduces another key event type called CONVERSION_EVENT, designed to efficiently identify and log the source of a conversion when it occurs. When the application detects that a conversion has taken place—such as when the REGISTRATION_SUCCESS screen is displayed or a relevant backend response is received—it first sends a regular event corresponding to this context. Immediately following this, the application emits a special CONVERSION_EVENT, which instructs the analytics framework to initiate a backtracking process within the corresponding client's event list.

```
EventName: CONVERSION_EVENT
SourceEventID: <eventid>
TargetEventID: <eventid>
```

The CONVERSION_EVENT contains two critical fields: targetEventId and sourceEventId. The targetEventId is filled by the application and refers to the event ID of the just-occurred conversion event. The sourceEventId, on the other hand, is initially left blank by the application. When the CONVERSION_EVENT is processed by the analytics framework or cache, the framework begins a trace-back operation by reading the name of the conversion event associated with the targetEventId. It then searches through the linked sequence of events for a corresponding BEACON_EVENT that matches this conversion event name as its target.

Upon finding the appropriate BEACON_EVENT, the framework retrieves the sourceEventId from it, which represents the entry point where the user's journey towards the conversion began. The analytics framework then populates this sourceEventId into the CONVERSION_EVENT and stores it in the cache. This completed event, now containing both the targetEventId and the sourceEventId, provides a clear and comprehensive record of the conversion, including both the outcome and its origin.

When this CONVERSION_EVENT is eventually uploaded to the backend, it offers precise insights into the conversion process: it identifies the specific conversion (e.g., REGISTERED_USER) along with the details contained in the targetEventId, and it also pinpoints the source entry point using the sourceEventId where the user's journey began.

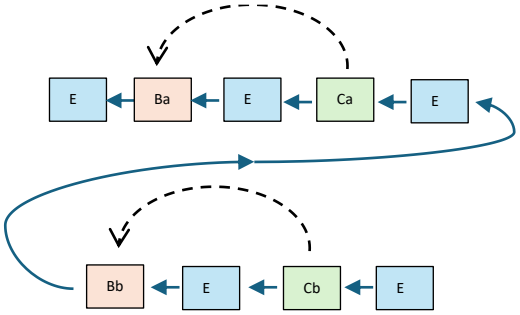
In cases where the corresponding BEACON_EVENT has already been uploaded and is no longer available in the local cache, the CONVERSION_EVENT is stored and uploaded without a sourceEventId. The backend, upon receiving such an event, recognizes the missing sourceEventId and performs the trace-back operation within its database. This backend process involves searching through the event sequence, event by event, moving backwards from the targetEventId to find the relevant BEACON_EVENT and thus determine the sourceEventId. To avoid excessive computational load, this backtracking process is limited to a fixed time window, such as the previous 24 hours of events. This time-bounded search ensures efficiency while maintaining the accuracy and reliability of conversion tracking, even in scenarios where the client's cache has already been cleared.

Examples

Use Case: Independent Conversion Paths

When an application has independent conversion paths, such as A -> B and C -> D, the instrumentation process is

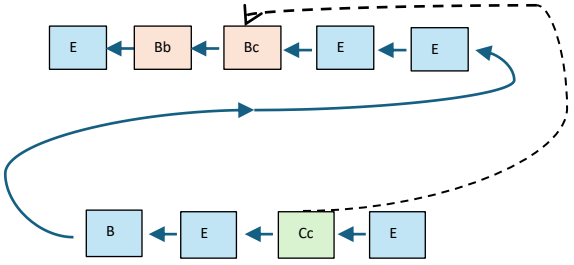
straightforward. The application emits one BEACON_EVENT at the starting point of each path (i.e., at points A and C) and a CONVERSION_EVENT at the end of each path (i.e., at points B and D). When a conversion event occurs, the analytics framework traces back through the event sequence to find the corresponding BEACON_EVENT. For example, when a user completes the conversion at point B, the framework will trace back to the BEACON_EVENT emitted at point A, effectively linking the conversion to its origin.



Use Case: Single Entry Point with Multiple Conversions

In scenarios where multiple conversions can originate from the same entry point, such as A -> B and A -> C, the application must account for all possible conversion paths from that origin. When the user interacts with the origin point A, the application emits a regular event for A, followed by multiple BEACON_EVENTs—one for each potential conversion path (Bb for B and Bc for C). If the user eventually completes the conversion at point C, the application will emit a regular event indicating the user's action at C, followed by a CONVERSION_EVENT with only the targetEventId filled in.

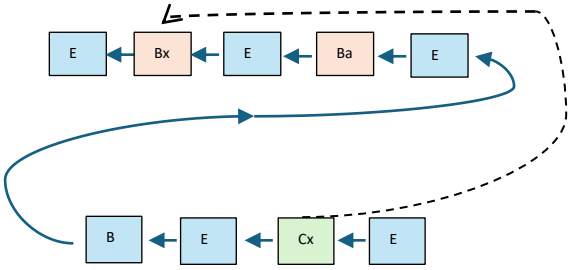
The analytics framework then performs a backtracking operation to discover the corresponding beacon event for this conversion, specifically searching for Bc and not Bb, as B did not occur. Upon finding Bc, the framework updates the CONVERSION_EVENT with the sourceEventId derived from Bc and stores this enriched event in the cache. This process ensures that the framework accurately traces the conversion back to its correct origin, even when multiple conversions share the same entry point.



Use Case: Mixed Conversion Paths

Consider a more complex scenario where the application has mixed conversion paths, such as A -> B -> C -> D and X -> B -> C -> Z, where UI elements B and C are part of both conversion paths. In this case, when the user visits point A, the application emits a BEACON_EVENT for the potential conversion ending at D. Similarly, when the user visits point X, another BEACON_EVENT is emitted for the potential conversion ending at Z.

As the user progresses through the shared UI elements B and C, these elements do not need to handle any specific conversion-related instrumentation. The focus remains on the initial beacons set at points A and X. When the user completes a conversion at either D or Z, the application emits the corresponding CONVERSION_EVENT, and the analytics framework performs a beacon search to identify the correct source event. The framework traces back from the conversion point to find the beacon event associated with the user's entry point, linking the conversion to its origin. This approach simplifies the instrumentation of shared UI elements, reducing the complexity and ensuring that the conversion tracking remains accurate and efficient.



Conclusion

The proposed design offers a highly efficient solution for tracking conversions in mobile applications, addressing the complexities associated with traditional methods that require the propagation of information through extensive instrumentation. By leveraging the analytics framework's event linking and beacon event mechanisms, this approach significantly reduces the need for intricate and error-prone instrumentation across multiple screens or user interactions. The introduction of BEACON_EVENTs at strategic points within conversion paths allows the system to accurately identify the origin of a conversion without requiring detailed tracking information to be carried forward through each step of the user journey.

This design ensures that applications can maintain clean, maintainable codebases while still achieving precise and reliable conversion tracking. The use of the analytics framework's cache to store and link events chronologically provides a robust foundation for backtracking and identifying source events. Even in complex scenarios with mixed or parallel conversion paths, the framework can efficiently determine the origin of any conversion, ensuring that product owners and analysts have access to critical insights regarding user behavior and engagement.

Moreover, the flexibility of this design allows it to handle corner cases where links between events may be broken or when certain events are missing from the local cache. The backend's ability to perform a limited trace-back search ensures that conversions can still be accurately tracked, even in less-than-ideal conditions. Overall, this approach not only enhances the accuracy and efficiency of conversion tracking but also simplifies the overall process, making it easier to implement and maintain across a wide range of applications. By eliminating the need for cumbersome and often unreliable instrumentation, this design represents a significant advancement in mobile analytics, providing a scalable and effective solution for understanding user interactions and optimizing conversion rates.

```
// Pseudo code for adding event into cache in framework with back trace logic
// Class representing the Analytics Framework
class AnalyticsFramework {

  // Cache to store events
  cache = []

  // Function to add an event to the framework
  function addEvent(Event event) {

   // process given Event : updating framework relevant info & update prevEventId
   processEvent(event);
   // Check if the event is of type CONVERSION_EVENT
    if (event.type == "CONVERSION_EVENT") {
      // Start the trace back process
      targetEventName = getEventName(event.targetEventId)
      previousEventId = event.previousEventId

      // Initialize sourceEventId to null
      sourceEventId = null

      // Traverse back through the previous events in the cache
      while (previousEventId != null) {
        // Get the previous event from the cache
        previousEvent = cache.getEvent(previousEventId)

        // If the previous event is a BEACON_EVENT and matches the target event name
        if (previousEvent.type == "BEACON_EVENT" && previousEvent.targetEventName == targetEventName) {
          // Set the sourceEventId to the ID of this BEACON_EVENT
          sourceEventId = previousEvent.id
          break
        }

        // Move to the next previous event in the chain
        previousEventId = previousEvent.previousEventId
      }

      // If a corresponding beacon event was found, update the CONVERSION_EVENT with the sourceEventId
      if (sourceEventId != null) {
        event.sourceEventId = sourceEventId
      }

      // Add the CONVERSION_EVENT to the cache
      cache.add(event)
    } else {
      // For all other event types, simply add them to the cache
      cache.add(event)
    }

   // update local track of previous Event
   updatePreviousEvent(event);
  }

  // Helper function to get the name of an event given its ID
  function getEventName(EventId eventId) {
    event = cache.getEvent(eventId)
    return event.name
  }
```

References

[1] https://www.singular.net/glossary/app-analytics/
[2] https://amplitude.com/guides/mobile-analytics
[3] https://hyperight.com/data-and-analytics-trends-that-will-loom-large-in-2021-and-beyond/
[4] https://www.smartlook.com/blog/trends-analytics-2021/
[5] https://www.acumenresearchandconsulting.com/data-analytics-market