Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Studio e implementazione di un sistema per la tracciabilità dell'utilizzo di Android Auto basato su IOTA

Relatore:
Dott. Luca Sciullo

Correlatore:
Dott. Lorenzo Gigli

Presentata da:
Alexandru Bogdan Nicolescu

Sessione II
Anno Accademico 2023/2024

Monitoraggio Attività

Guida Sicura

IOTA

Distributed Ledger Technology

Android Auto

A mia madre e a tutti
coloro che hanno
creduto in me

Abstract

Gli incidenti stradali rappresentano una delle principali cause di morte a livello globale, spesso causati dalle distrazioni dei conducenti, fra queste l'interazione con dispositivi come smartphone. Questa tesi propone un sistema innovativo per il monitoraggio delle attività su Android Auto, sviluppato attraverso un'applicazione Android denominata Blackbox. L'obiettivo è registrare e archiviare in modo sicuro i dati raccolti utilizzando la tecnologia Distributed Ledger Technology (DLT) di IOTA. Questo sistema apre prospettive interessanti per il suo utilizzo nel mondo assicurativo. Ad esempio, i dati raccolti potrebbero essere impiegati per analizzare il rischio associato a determinati stili di guida o per determinare la responsabilità in caso di sinistri.

Il sistema sfrutta l'API UsageStatsManager per monitorare eventi rilevanti, come cambi di applicazione e notifiche, nonché l'architettura Tangle di IOTA per garantire sicurezza, trasparenza e scalabilità senza costi di transazione. L'implementazione include un modulo per la raccolta e il filtraggio degli eventi di utilizzo, la gestione locale delle informazioni e il loro trasferimento sicuro sulla rete distribuita tramite un processo di Proof of Work. I risultati ottenuti evidenziano la fattibilità del sistema, sebbene limitazioni intrinseche all'API scelta impongano ulteriori sviluppi per una rilevazione più precisa delle interazioni pericolose.

Indice

Capitolo 1

Introduzione

Gli incidenti stradali rappresentano una delle principali cause di morte e feriti a livello globale, con milioni di vittime ogni anno [18]. Tra i fattori di rischio più critici vi sono la guida distratta e l'uso improprio dello smartphone durante la guida [8]. Negli ultimi anni, l'attenzione della ricerca si è concentrata sul mitigare questi rischi.

Android Auto è una piattaforma progettata per integrare i dispositivi mobili con i sistemi di infotainment dei veicoli, rappresenta un punto d'incontro tra tecnologia e guida sicura [7]. Tuttavia, l'uso di questa tecnologia comporta il rischio di distrazioni per i conducenti [9], che potrebbero avere conseguenze gravi. Esiste quindi la necessità di strumenti in grado di monitorare e registrare l'interazione con tali tecnologie.

Questo progetto di tesi si propone di sviluppare un sistema per la detection delle attività svolte tramite Android Auto, utilizzando un'applicazione Android per raccogliere dati sugli eventi d'uso e registrarli in modo sicuro e immutabile su una distributed ledger technology (DLT) [4].

Il sistema proposto in questa tesi non ha lo scopo di prevenire direttamente gli incidenti stradali, ma fornisce uno strumento per analizzarli. Ad esempio, nel contesto assicurativo, l'applicazione potrebbe essere utilizzata per verificare se il conducente fosse distratto al momento di un incidente, monitorando le interazioni con Android Auto. Questo approccio garantirebbe una valutazione più oggettiva delle responsabilità, supportata da registrazioni immutabili archiviate sulla DLT.

Il lavoro si basa sull'idea di combinare tecnologie già esistenti per sviluppare il sistema proposto. L'uso delle API di Android, in particolare di `UsageStatsManager`, permette di raccogliere informazioni sugli eventi di utilizzo. Tuttavia, queste API presentano limitazioni che saranno discusse nel capitolo 5. La DLT di IOTA [16] è una soluzione

per garantire la sicurezza, l'integrità e la trasparenza dei dati raccolti e, grazie alla sua architettura, consente transazioni rapide e prive di costi.

Il documento si articola in cinque capitoli: il primo è lo **stato dell'arte** dove sarà trattato il contesto tecnologico attuale con un focus su Android Auto, la detection di attività durante la guida, le tecnologie DLT con blockchain e IOTA con il loro utilizzo nel contesto della mobilità, il secondo capitolo, **progettazione del sistema**, dove verrà introdotta l'architettura del sistema discutendo le scelte progettuali e descrivendo il flusso operativo, il capitolo dell'**implementazione** approfondisce l'implementazione del sistema dando una panoramica dell'applicazione, un'introduzione alle tecnologie utilizzate e descrivendo i moduli principali, in **valutazioni e risultati** si presentano i test effettuati per analizzare le prestazioni, i costi e i limiti del sistema proposto, infine in **conclusioni e sviluppi futuri** saranno riassunti i risultati principali evidenziando le potenzialità di miglioramento e applicazioni future.

Capitolo 2

Stato dell'arte

2.1 Android Auto

Android Auto[7] è una piattaforma progettata per migliorare l'interazione tra il conducente e il sistema di infotainment del veicolo, riducendo le distrazioni e consentendo un utilizzo più sicuro delle applicazioni del proprio smartphone durante la guida. Attraverso una connessione tra un dispositivo Android e il sistema di bordo del veicolo, il conducente può accedere in modo semplificato a funzionalità essenziali come la navigazione, la gestione di chiamate o messaggi, la riproduzione musicale e altre applicazioni supportate.

L'architettura di Android Auto si basa su una stretta integrazione tra hardware e software, garantendo che le applicazioni siano ottimizzate per l'utilizzo durante la guida. Essa si compone di diverse componenti. Il **Dispositivo Android** è responsabile dell'elaborazione principale: esegue le applicazioni compatibili, gestisce i dati provenienti da sensori e GPS e invia il contenuto al sistema di infotainment del veicolo. Il **Sistema di infotainment del veicolo** funge da interfaccia utente, presentando le informazioni tramite uno schermo touchscreen o consentendo il controllo tramite pulsanti fisici e comandi vocali. La comunicazione tra il dispositivo e il sistema del veicolo avviene attraverso i **Protocolli di connessione**, che includono USB, Bluetooth o in alcune configurazioni più recenti, il Wi-Fi. Infine, i **Servizi Google** gestiscono funzionalità come Google Maps per la navigazione, l'assistente vocale per i comandi vocali e aggiornamenti in tempo reale per traffico e meteo.

Android Auto offre un'interfaccia minimalista e intuitiva, progettata per essere semplice e sicura durante la guida. L'interfaccia visiva si adatta al display del sistema di infotainment del veicolo e utilizza elementi chiari e ben visibili, minimizzando l'interazione necessaria. Inoltre, è possibile utilizzare comandi vocali per controllare applicazioni e funzioni, riducendo al minimo la distrazione visiva e manuale. Le applicazioni compati-

bili includono servizi di navigazione come Google Maps e Waze, piattaforme di streaming musicale come Spotify e YouTube Music, oltre a strumenti per la comunicazione come WhatsApp e Messaggi.

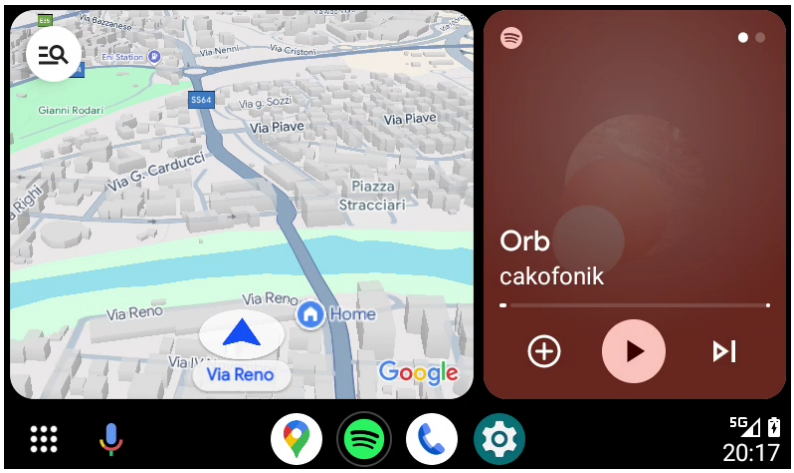

Figura 2.1: Interfaccia Android Auto.

Grazie alla sua architettura e alle sue funzionalità, Android Auto rappresenta un sistema versatile e sicuro per migliorare l'esperienza di guida. Tuttavia, per garantire la sicurezza stradale rimane fondamentale che il conducente limiti l'interazione con il sistema ai comandi strettamente necessari, affidandosi il più possibile alle funzionalità vocali e automatizzate.

2.2 Detection di attività alla guida

Come riportato dallo studio di Voinea et al.[18] gli incidenti stradali causano più di 1,35 milioni di morti e 50 milioni di feriti all'anno in tutto il mondo e una causa significativa di tali incidenti è la guida distratta. La detection di attività alla guida è un argomento d'interesse da molti anni ed è stato affrontato da molti punti di vista in contesti diversi. In questa sezione vedremo alcuni studi che affrontano il problema in diversi modi.

Lo studio di Xing et al.[19] affronta il problema della distrazione dei conducenti, motivata dalla crescente integrazione della tecnologia di bordo e dal corrispondente aumento delle attività "secondarie". La ricerca mira a migliorare la sicurezza del traffico identificando e analizzando con precisione le posture e le attività svolte dal conducente. Utilizzando un sensore Kinect (una telecamera RGB), i ricercatori hanno sviluppato un sistema per riconoscere in tempo reale sia le attività di guida standard sia i compiti secondari. Sono stati identificati sette compiti: quattro associati alla guida normale (ad esempio il controllo dello specchietto) e tre secondari, potenzialmente distraenti (ad esempio

l'invio di messaggi o l'uso del cellulare). La metodologia prevedeva l'acquisizione di dati multimodali, tra cui le rotazioni 3D della testa e le posizioni delle articolazioni della parte superiore del corpo che sono stati elaborati e analizzati per rilevare i comportamenti del conducente. Una rete neurale feedforward (FFNN) ha dimostrato un'accuratezza superiore all'80% nel riconoscimento di questi compiti, sottolineando l'importanza di combinare i dati sulla postura della testa e del corpo per una classificazione precisa. Lo studio sottolinea la necessità di monitorare il conducente in tempo reale per migliorare la sicurezza di guida sia per i veicoli convenzionali che per quelli semi-autonomi, dove il conducente è autorizzato a togliere le mani dal volante.

Nel contesto della guida autonoma condizionale, il riconoscimento delle attività del conducente riveste un ruolo cruciale per valutare la sua prontezza a riprendere il controllo del veicolo in situazioni specifiche. Lo studio di Braunage et al.[2] esplora un'architettura che utilizza dati di tracciamento della testa e degli occhi per automatizzare il riconoscimento delle attività del conducente, con particolare attenzione alle attività secondarie svolte durante la guida autonoma.

L'approccio integra i movimenti della testa e degli occhi per identificare le attività, evitando di utilizzare stime, spesso inaffidabili, della direzione dello sguardo. Utilizzando dati raccolti da uno studio in simulatore di guida con 73 partecipanti, classificandoli con un modello di machine learning supervisionato di tipo support-vector machine. Il framework analizza comportamenti come saccadi (movimenti rapidi degli occhi volti a focalizzare l'obiettivo), fissazioni e rotazioni della testa. Le attività secondarie esaminate includono la lettura, la visione di video, la scrittura di e-mail e l'inattività. I risultati mostrano che le attività secondarie, in particolare quelle che richiedono un elevato coinvolgimento visivo o manuale, influenzano negativamente la qualità della ripresa del controllo da parte del conducente. Le nuove caratteristiche derivanti dai modelli di movimento della testa e dalle saccadi hanno migliorato significativamente la precisione nella classificazione delle attività rispetto alle tecniche tradizionali. Questa ricerca mette in evidenza il potenziale delle tecnologie avanzate di sensori e dei modelli di apprendimento automatico per migliorare la sicurezza nei contesti di guida autonoma.

Uno studio condotto da Ohn-Bar et al.[14] ha proposto un framework multimodale per il riconoscimento delle attività del conducente che sfrutta le posizioni di testa, occhi e mani per classificare le azioni compiute dal conducente. Questo approccio utilizza una configurazione di due telecamere per monitorare simultaneamente i movimenti delle mani e della testa. Queste informazioni sono usate per classificare le attività tramite l'utilizzo di modelli SVM, un modello per determinare la posizione delle mani, la postura della testa e la direzione dello sguardo e un altro che combina queste informazioni per determinare l'attività. Il sistema è stato valutato in ambienti di guida reali, classificando tre tipologie principali di attività: interazione con il volante (entrambe le mani sul volante), interazione con il cambio e attività nella zona del quadro strumenti. Integrando i dati provenienti

dai movimenti delle mani con quelli della testa e degli occhi, il sistema è stato in grado di migliorare significativamente la precisione del riconoscimento delle attività solamente rispetto all'uso della posizione delle mani. Questo approccio dimostra il potenziale di una visione multimodale per affrontare le sfide del monitoraggio del conducente.

Un contributo innovativo è dato da Fan et al.[5] con il sistema "SafeDriving". Questo utilizza segnali elettromiografici (EMG) per identificare comportamenti di guida anomali. Questo approccio si basa sull'acquisizione di segnali EMG tramite un sensore indossabile posizionato sull'avambraccio del conducente, progettato per catturare segnali elettrici muscolari associati a movimenti specifici. I comportamenti anomali monitorati includono movimenti come chinarsi per raccogliere oggetti, toccare il tettuccio, sterzare bruscamente e girarsi indietro. L'analisi di questi segnali avviene attraverso una rete neurale di tipo Gated Recurrent Unit (GRU), che grazie alla capacità di tenere conto delle dipendenze temporali, ha raggiunto un'accuratezza media del 93,94% nella classificazione di questi comportamenti. SafeDriving si distingue per la sua capacità di fornire una rilevazione accurata in tempo reale, utilizzando sensori relativamente economici e un sistema basato su smartphone. Lo smartphone è responsabile della suddivisione dei segnali EMG in frame, dell'estrazione delle feature e delle previsioni con la rete GRU.

Un lavoro rilevante in questo campo che ha alcuni aspetti in comune con questo progetto è quello presentato da Gerrits et al.[6], che propone un'applicazione per Android Automotive progettata per rilevare e registrare dati in caso di incidenti stradali. L'applicazione utilizza la piattaforma Android Automotive OS per raccogliere informazioni dai sensori del veicolo come velocità, accelerazione, posizione GPS e dati generati da oltre 130 sensori integrati nel veicolo. L'obiettivo principale è garantire che i dati siano raccolti in tempo reale e archiviati in modo sicuro e immutabile tramite una blockchain privata e il sistema decentralizzato IPFS (InterPlanetary File System).

La proposta degli autori prevede che, in caso di incidente, i dati siano acquisiti dall'applicazione e immediatamente inviati al sistema IPFS. Tuttavia, per evitare ritardi potenziali legati alla connessione, il Content Identifier (CID), che rappresenta un hash univoco utilizzato per recuperare i dati da IPFS, viene generato localmente all'interno del veicolo prima di essere memorizzato sulla blockchain privata basata su Substrate (un framework per la creazione di blockchain ad-hoc). Questo approccio permette di garantire efficienza e tempestività nella gestione delle informazioni critiche in situazioni di emergenza. Inoltre, l'architettura utilizza Protobuf per la serializzazione dei dati, ottimizzando la trasmissione e la memorizzazione di grandi quantità di informazioni. Sebbene il focus di quel lavoro sia orientato agli incidenti stradali, i principi alla base della loro implementazione, come la raccolta di dati in real time e la memorizzazione sicura su blockchain, rappresentano una base solida per lo sviluppo di sistemi simili che monitorano l'uso di Android Auto in contesti differenti.

2.3 Uso dello smartphone durante la guida

Come riportato da Guo et al.[8] la distrazione da cellulare è uno dei principali fattori che contribuiscono agli incidenti stradali, una delle principali cause di morte in tutto il mondo. Inoltre, nell'articolo è riportato che l'8% degli incidenti stradali negli Stati Uniti è causato dall'uso dello smartphone durante la guida. In questa sezione saranno discussi gli studi riguardo alla detection dell'uso dello smartphone durante la guida e di come questo impatti il conducente.

Un contributo significativo al tema è offerto dallo studio complementare condotto da Khan et al.[9], lo studio analizza le distrazioni causate dall'uso dello smartphone durante la guida utilizzando il dataset generato da un'applicazione chiamata "AutoLog". Questa applicazione sfrutta i sensori integrati negli smartphone e un modulo ELM327 collegato alla porta OBD2 del veicolo per raccogliere dati relativi alle attività dello smartphone, alla dinamica del veicolo e a fattori ambientali.

Lo studio evidenzia che attività come la messaggistica o la gestione delle chiamate durante la guida influenzano significativamente il comportamento del conducente portando a conseguenze dirette sulla dinamica del veicolo. Ad esempio si osservano variazioni improvvise della velocità durante una telefonata; ciò aumenta il rischio di incidenti su strade trafficate. Inoltre sono comuni deviazioni di corsia causate da cambiamenti repentini dell'angolo di sterzata, spesso registrate durante interazioni continue con lo smartphone (scrittura di messaggi). Altri effetti osservati includono frenate brusche che compromettono la sicurezza del veicolo e infine si è riscontrato un evidente sovraccarico cognitivo derivato dall'utilizzo dello smartphone che riduce notevolmente la capacità di reazione del conducente agli imprevisti. Lo studio sottolinea che le interfacce native degli smartphone non sono adeguate per un utilizzo sicuro durante la guida poiché richiedono un'elevata attenzione. Per migliorare la sicurezza stradale e ridurre le distrazioni, è fondamentale sviluppare interfacce capaci di adattarsi al contesto di guida.

Un altro interessante approccio alla rilevazione dell'uso dello smartphone durante la guida è descritto nello studio di Berri et al.[1] che presenta un sistema di riconoscimento di pattern basato sull'analisi delle immagini catturate da una telecamera frontale posizionata sul cruscotto del veicolo. Il sistema utilizza questa telecamera per identificare il telefono nelle mani del conducente attraverso il rilevamento della pelle e altre feature visive. I dati elaborati sono quindi classificati tramite una support-vector machine (SVM) con kernel polinomiale, che ha ottenuto un'accuratezza media del 91,57% sulle immagini analizzate. Questo approccio dimostra come tecniche avanzate di elaborazione delle immagini possano essere utilizzate per monitorare in tempo reale il comportamento del conducente, contribuendo a migliorare la sicurezza stradale.

Lo studio di Montori et al.[12] offre un approccio innovativo basato sull'uso della fotocamera frontale degli smartphone per rilevare l'uso del dispositivo durante la guida.

L'approccio proposto supera i limiti di tecniche basate su sensori inerziali che sono soggetti a potenziali errori e adotta metodi di classificazione delle immagini e rilevamento degli oggetti. Il sistema utilizza la fotocamera per identificare il posizionamento del conducente o del passeggero, sfruttando caratteristiche distintive come l'orientamento della cintura di sicurezza e la posizione del finestrino. Dopo un'iniziale classificazione istantanea, i ricercatori hanno implementato una classificazione continua su sequenze di immagini, ottenendo un'accuratezza di circa 90% in pochi secondi. Questo risultato pone le basi per una potenziale integrazione in applicazioni o sistemi operativi per smartphone per la rilevazione in tempo reale della distrazione alla guida senza necessità di hardware aggiuntivo.

2.4 Distributed Ledger Technology

Come spiegato da Ioini et al.[4] le distributed ledger technology (DLT) hanno guadagnato un'importante attenzione, soprattutto per la loro capacità di facilitare interazioni sicure e decentralizzate senza la necessità di fidarsi di un intermediario. Di base sono strutture dati per registrare le transazioni e un insieme di funzioni per manipolarle. Le DLT si basano su tecnologie come la crittografia a chiave pubblica per stabilire un'identità digitale per ogni partecipante, le reti peer-to-peer distribuite per evitare un single point of failure e i meccanismi di consenso per consentire a tutti i partecipanti di concordare su un'unica versione di verità, senza il bisogno di una parte terza. Le DLT discusse nell'articolo sono quattro e differiscono per la loro struttura.

La blockchain, la DLT più conosciuta, funziona come una lista concatenata di blocchi. Ogni blocco contiene un insieme di transazioni validate tramite un meccanismo di consenso. Il punto di forza principale della blockchain risiede nella sua immutabilità e trasparenza, sebbene presenti sfide legate alla scalabilità e ai costi di transazione.

Introdotto da IOTA, Tangle utilizza un Grafo Aciclico Diretto (DAG) invece di una catena lineare. Elimina le commissioni di transazione e consente la scalabilità, permettendo a ogni utente di validare le transazioni, rendendolo particolarmente adatto per applicazioni IoT.

Hashgraph utilizza una struttura DAG combinata con un protocollo di gossip e votazione per il consenso. Vanta alta efficienza e velocità nelle transazioni, anche se è ancora in fase sperimentale ed è brevettata.

Le architetture sidechain utilizzano una blockchain consorziata per gestire gli accessi a tutti i partecipanti e una serie di sidechain (blockchain private) per gestire le transazioni locali. Questo modello migliora la privacy e il controllo sullo scambio di dati, ma è limitato a determinati contesti di rete.

2.5 Blockchain

Come spiegato da Nofer et al.[13] la **blockchain** è una tecnologia che permette di gestire dati in un registro distribuito garantendo sicurezza, trasparenza e immutabilità. Pur essendo nata come tecnologia alla base delle criptovalute, come il Bitcoin, le sue applicazioni si estendono a numerosi altri ambiti come la gestione dei dati, gli smart contract e i sistemi di tracciabilità. A differenza dei sistemi centralizzati dove l'affidabilità dipende da intermediari, la blockchain elimina la necessità di fidarsi di terze parti, trasferendo questa fiducia a un sistema matematico crittograficamente sicuro. Questo cambiamento di paradigma consente di ridurre significativamente i rischi associati a errori umani o alla malafede degli intermediari.

Dal punto di vista funzionale la blockchain si basa su una catena di blocchi, ciascuno dei quali contiene **dati transazionali** che rappresentano le informazioni registrate nel blocco, **timestamp** ovvero il momento in cui è stato generato il blocco, **hash crittografico**, **hash del blocco precedente** e **nonce** che è un valore casuale utilizzato nel processo di validazione dell'hash. Ogni blocco è identificato da un hash crittografico unico che dipende dai dati contenuti nel blocco e dal riferimento al blocco precedente, garantendo così l'integrità dell'intera catena. Grazie a questo meccanismo ogni tentativo di alterazione di un blocco richiederebbe la modifica di tutti i blocchi successivi, un'impresa praticamente impossibile senza il controllo della maggioranza dei nodi della rete. Il primo blocco della catena è noto come *genesis block* e rappresenta l'origine del registro.

Un elemento chiave della blockchain è il processo di **consenso**, che permette a una rete distribuita di nodi di accordarsi sullo stato del registro senza la necessità di un intermediario. Secondo Swanson[17] il consenso è *"il processo mediante il quale una maggioranza (o in alcuni casi, tutti) dei validatori della rete concordano sullo stato di un registro"*. Tra i principali algoritmi di consenso troviamo: **Proof-of-Work (PoW)** dove i nodi competono per risolvere complessi problemi matematici, validando le transazioni e ricevendo ricompense (ad esempio Bitcoin utilizza PoW) e **Proof-of-Stake (PoS)** dove i validatori vengono selezionati in base alla quantità di asset posseduti, riducendo il consumo energetico rispetto al PoW (ad esempio Ethereum dal 2022 usa PoS).

Le applicazioni della blockchain si estendono oltre le criptovalute. Come evidenziato da Nofer et al.[13] questa tecnologia può offrire benefici a settori come la **finanza** con la gestione di valute digitali, emissione di titoli e sistemi di pagamento, il settore **notarile** dove la registrazione e verifica di proprietà intellettuale o documenti ufficiali non necessitano più dell'autorizzazione centrale del notaio, il settore **musicale** con la determinazione delle royalties musicali e gestione della proprietà dei diritti musicali, il settore dell'**internet of things (IoT)** con la gestione delle comunicazioni in maniera affidabile tra dispositivi intelligenti e il settore **logistico** con sistemi per monitorare l'origine e il percorso di beni fisici, come cibo o medicinali. Inoltre, con l'introduzione degli

smart contract, la blockchain consente di automatizzare e rendere più efficienti i processi contrattuali, riducendo la necessità di intermediari e abbassando i costi operativi.

Nonostante i numerosi vantaggi, la blockchain presenta anche delle sfide. La scalabilità è uno dei principali problemi, poiché l'aumento del numero di transazioni può rallentare il sistema e aumentare i costi energetici. Anche la privacy rappresenta un'area critica, poiché l'immutabilità del registro può entrare in conflitto con il diritto all'oblio e con la protezione dei dati personali.

Secondo gli autori, questa innovazione potrebbe trasformare modelli di business esistenti, crearne di nuovi e influenzare intere industrie, rendendo essenziale lo studio delle interazioni tra tecnologia, mercati e modelli economici.

2.6 IOTA

Come spiegato da Silvano et al.[16] **IOTA** è una criptovaluta sviluppata per superare le limitazioni delle blockchain tradizionali, cioè quelle di essere "decentralizzate", "sicure" e "scalabili" allo stesso tempo; le blockchain tradizionali possono avere solo due di queste proprietà e questo è noto come "*trilemma della scalabilità*". IOTA offre scalabilità, trasferimenti veloci e privi di costi, con il focus sull'internet of things. L'IoT rappresenta un vasto sistema globale di dispositivi eterogenei e decentralizzati che scambiano dati tramite protocolli di comunicazione standardizzati. La sua architettura è denominata Tangle, rappresenta una tipologia di grafo aciclico diretto (DAG), che elimina la necessità di blocchi sequenziali e miner, rendendo la rete altamente scalabile, priva di costi di transazione e con tempi di trasferimento quasi istantanei.

A differenza di sistemi come Bitcoin ed Ethereum che utilizzano una blockchain con blocchi sequenziali contenenti più transazioni, IOTA adotta una struttura diversa basata su un **DAG** chiamato **Tangle**. Nel Tangle non esistono blocchi, ogni nuova transazione fa riferimento a due transazioni precedenti. Quando un partecipante vuole aggiungere una transazione deve approvare due transazioni precedenti, certificando implicitamente che sono valide, così come tutti i loro predecessori. Questa modalità elimina la dipendenza dai miner, riducendo sia i costi energetici che i requisiti computazionali. In particolare si hanno **transazioni senza costi** dove il costo associato a una transazione è limitato alla potenza computazionale necessaria per validare altre due transazioni, rendendo la rete ideale per microtransazioni economiche, **scalabilità** dove l'architettura permette che il numero di transazioni processate aumenti proporzionalmente al numero di utenti, grazie alla possibilità di eseguire transazioni in parallelo, **efficienza energetica** ovvero il Tangle richiede meno energia rispetto alle blockchain tradizionali, rendendolo adatto ai dispositivi IoT con risorse limitate.

Un'altra caratteristica di IOTA è il protocollo **Masked Authenticated Messaging (MAM)**, che consente la trasmissione sicura e crittografata di dati tra dispositivi IoT. MAM utilizza firme crittografiche basate su alberi di hash Merkle e offre modalità operative pubbliche, private e ristrette, garantendo un elevato livello di sicurezza e controllo sull'accesso ai dati.

Una parte essenziale del funzionamento di IOTA è garantire l'integrità e la sicurezza registrate nel Tangle e come viene riportato da Raman et al.[15] a questo scopo svolge un ruolo cruciale la funzione di hash **Curl-P**. Questa è una funzione crittografica progettata per generare hash di dimensione fissa a partire da input di lunghezza variabile, fornendo protezione contro le manomissioni dei dati. Questo è particolarmente importante nel contesto delle supply chain, dove la trasparenza e l'immutabilità dei dati sono fondamentali per garantire la fiducia tra i vari attori coinvolti.

Curl-P è utilizzato per proteggere le transazioni registrate nel Tangle, assicurando che ogni transazione sia univocamente identificabile e che le informazioni ad essa associate non possano essere alterate senza invalidare il corrispondente hash. Questa proprietà rende il sistema resistente a tentativi di frode e manomissione, poiché qualsiasi modifica non autorizzata dei dati sarebbe immediatamente rilevabile.

Dal punto di vista tecnico Curl-P viene implementato insieme ad altri meccanismi di sicurezza, come il *Proof of Work* (PoW) e l'algoritmo di selezione dei tip basato su *Markov Chain Monte Carlo* (MCMC). Questi componenti lavorano insieme per garantire una validazione sicura delle transazioni.

2.7 Confronto tra Blockchain e IOTA

La blockchain e IOTA rappresentano due tecnologie complementari e allo stesso tempo alternative, progettate per gestire in modo sicuro i dati in ambienti distribuiti. Come discusso da Khrais et al.[11], entrambe le tecnologie si concentrano sull'integrità e sull'affidabilità dei dati, ma adottano approcci differenti per affrontare problemi come la scalabilità, i costi di transazione e il consumo energetico.

Le blockchain come Bitcoin o Ethereum, utilizzano una struttura lineare e sequenziale in cui i blocchi sono collegati in una catena. Ogni blocco contiene un hash del blocco precedente, il che garantisce l'integrità dei dati. In questo sistema, il consenso è ottenuto tramite algoritmi come il Proof-of-Work (PoW), che richiedono una significativa potenza computazionale. IOTA invece, adotta una struttura di grafo aciclico diretto (DAG) chiamata Tangle, dove non ci sono blocchi né miners. Ogni transazione valida approva due transazioni precedenti, contribuendo alla decentralizzazione e alla scalabilità del sistema. Questo consente a IOTA di operare senza costi di transazione, un vantaggio cruciale per applicazioni IoT, dove le microtransazioni sono essenziali.

Le applicazioni delle due tecnologie riflettono le loro differenze fondamentali: **Blockchain** è ampiamente utilizzata in finanza, supply chain e smart contract, dove la sicurezza è prioritaria, **IOTA** è ideale per scenari IoT, come smart cities, gestione delle infrastrutture e microtransazioni tra dispositivi.

Nonostante le differenze, entrambe le tecnologie contribuiscono a costruire ecosistemi affidabili e decentralizzati. La scelta tra blockchain e IOTA dipende dal caso d'uso specifico e dai requisiti di scalabilità e costo.

2.8 DLT nel contesto della mobilità

Gli utilizzi delle DLT nel contesto della mobilità possono essere molteplici, un esempio è stato quello precedentemente discusso di Gerrits et al.[6]. In questa sezione saranno esplorati altri utilizzi.

Un contributo è quello di Butera et al.[3] che introduce un sistema decentralizzato basato su blockchain che utilizza la gamification per promuovere uno stile di guida sicuro. La piattaforma combina dati raccolti dai sensori degli smartphone come GPS e accelerometro con un sistema di punteggio gestito da uno smart contract presente sulla blockchain Ethereum. I guidatori accumulano punti evitando manovre pericolose classificate con un sistema basato su threshold e i migliori sono ricompensati con premi in token. Questo approccio garantisce trasparenza e integrità dei dati grazie all'immutabilità della blockchain e automatizza la distribuzione dei premi attraverso smart contract. L'uso delle DLT in questo contesto non solo incentiva la sicurezza stradale ma potrebbe anche essere integrato in polizze assicurative personalizzate, dimostrando il potenziale delle tecnologie decentralizzate nell'evoluzione dei servizi legati alla mobilità.

Un altro contributo è dato dallo studio di Khan et al.[10] che propone l'integrazione di tecnologie di deep learning e blockchain per rilevare comportamenti distratti durante la guida e garantire la trasmissione sicura di dati multimediali. La loro soluzione sfrutta una rete neurale convoluzionale (CNN) addestrata sul dataset "State Farm Distracted Driver Detection" per analizzare i video raccolti dalla telecamera interna al veicolo, individuando nove categorie di distrazioni: l'uso del telefono per chiamate, invio dei messaggi, consumo di bevande, sonnolenza, guida normale, interazione con il passeggero, regolazione della radio, guardare indietro e inattività. Il sistema ha raggiunto un'accuratezza del 86,02% nella rilevazione delle distrazioni. I video vengono frammentati in intervalli temporali fissi e criptati con l'algoritmo di hashing SHA-256 prima di essere registrati su una blockchain basata su Ethereum. Questa architettura consente di assicurare l'integrità dei dati trasmessi. Lo studio sottolinea il potenziale della fusione tra deep learning e blockchain per migliorare la sicurezza stradale.

Un lavoro che ha principi in comune con questo è presentato da Zichichi et al.[20] dove viene presentata la dApp chiamata "MOVO". Il sistema utilizza IOTA Tangle come DLT per garantire la memorizzazione immutabile e decentralizzata dei dati raccolti da veicoli e infrastrutture connesse. Attraverso l'applicazione Android, i dati generati dai sensori del veicolo vengono raccolti, elaborati e archiviati su IPFS (InterPlanetary File System), mentre i riferimenti ai dati sono registrati su canali MAM (Masked Authenticated Messaging) di IOTA. Questa combinazione garantisce una gestione sicura, scalabile e accessibile dei dati. Inoltre, MOVO integra smart contract sulla blockchain di Ethereum per automatizzare processi come l'accesso ai dati e i pagamenti relativi a servizi di mobilità, inclusi il monitoraggio delle condizioni del veicolo e la gestione delle ricariche per veicoli elettrici. Questo approccio dimostra il potenziale delle DLT non solo nella gestione dei dati ma anche nell'automatizzazione e nella sicurezza dei servizi di mobilità moderna.

Capitolo 3

Progettazione del sistema

3.1 Architettura

Il sistema progettato ha come obiettivo principale quello di monitorare l'utilizzo di Android Auto durante la guida, elaborare i dati raccolti e registrarli in modo sicuro su una DLT. Per fare ciò si è sviluppata un'applicazione Android denominata *Blackbox* che permette di registrare gli eventi di utilizzo di Android Auto e inviare le registrazioni sulla rete IOTA.

Per monitorare l'utilizzo di Android Auto è stata scelta l'API `UsageStatsManager`, anche se quest'API non è stata progettata specificamente per Android Auto, è abbastanza generica da poter rilevare anche il suo utilizzo. In assenza di soluzioni ufficiali, è stato scelto un approccio di polling che interroga periodicamente l'API per richiedere gli eventi avvenuti. Sebbene questa scelta sia limitata rispetto a un monitoraggio dettagliato come potrebbe offrire un'applicazione di sistema, offre comunque una panoramica abbastanza precisa degli eventi rilevanti come il cambio di app, notifiche o l'attivazione di un servizio in primo piano (foreground service), utili per la detection dell'interazione con Android Auto. Quindi, ogni volta che accade un'interazione come l'apertura di un'app, la ricezione di un messaggio, l'esecuzione di una telefonata o l'utilizzo dell'assistente vocale, essa viene registrata per poi essere inviata sulla DLT.

Per quanto riguarda la registrazione sicura dei dati su una DLT, si è scelto IOTA per i suoi bassi tempi di trasferimento e l'assenza di costi per le transazioni, dati dalla sua struttura *Tangle*. Inoltre, grazie alla sua funzione di hash Curl-P, garantisce la protezione contro le manomissioni dei dati, un requisito fondamentale in questo contesto. Altre blockchain affermate come Bitcoin o Ethereum avrebbero offerto maggiore longevità al sistema, in quanto IOTA è in continua evoluzione, ma il loro costo e i ritardi nella trasmissione, dati

dalla struttura sequenziale, non sarebbero state adeguate per l'alto numero di transazioni che deve effettuare il sistema.

L'elaborazione dei dati è stata progettata per avvenire sul dispositivo, riducendo la dipendenza dalla connessione di rete e aumentando l'affidabilità del sistema, poiché i dati possono essere raccolti, elaborati e memorizzati anche in assenza di connettività.

Una scelta chiave nell'architettura è stata l'integrazione del Proof of Work (PoW) direttamente nell'applicazione. Questo meccanismo è necessario per la validazione delle transazioni sulla rete IOTA ed è stato progettato per impattare al minimo sulle prestazioni generali del dispositivo. Questa scelta ha permesso di non utilizzare un backend separato per gestire l'interazione con la DLT, ma di implementare tutto il flusso di elaborazione e invio dei dati direttamente sull'applicazione. In questo modo si garantisce una latenza di trasmissione minima e l'integrità dei dati.

Infine, l'architettura è stata concepita in maniera modulare, permettendo di estendere e modificare facilmente il sistema in futuro. La separazione tra il monitoraggio e le varie fasi dell'invio dei dati permette di integrare nuove funzionalità, come nuovi protocolli DLT o metodi alternativi di detection delle attività, senza compromettere la stabilità del sistema.

Una panoramica dell'architettura è mostrata nella figura 3.1 la quale mostra come i diversi componenti interagiscono tra loro. L'utente avvia la registrazione e le sue interazioni con Android Auto (un'interfaccia che mostra le applicazioni del dispositivo) generano eventi; questi vengono registrati, salvati su un database locale e inviati alla DLT di IOTA, dove sono archiviati in modo sicuro e distribuito.

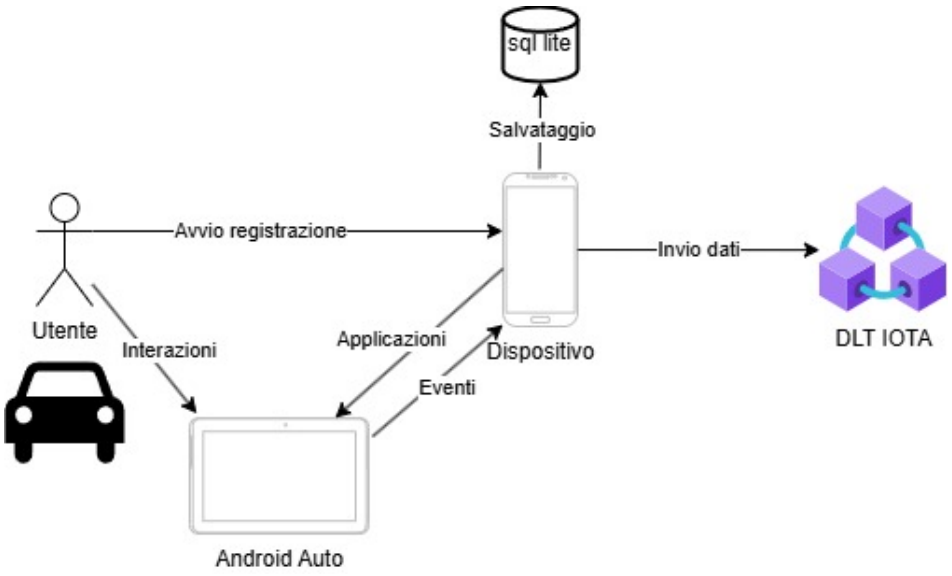

Figura 3.1: Architettura.

Il diagramma di sequenza mostrato in figura 3.2 descrive il processo di salvataggio delle registrazioni. Inizialmente i dati vengono archiviati localmente per consentire una futura visualizzazione. Successivamente, dopo aver completato il Proof of Work, le informazioni vengono inviate al nodo della rete IOTA per essere registrate in maniera sicura. Una volta che le informazioni sono state salvate, il nodo risponde con l'identificativo del blocco; una volta ricevuto quest'ultimo l'applicazione recupera la registrazione salvata localmente per aggiornarla con l'identificativo del blocco, così da poter visualizzare la registrazione tramite l'explorer della rete.

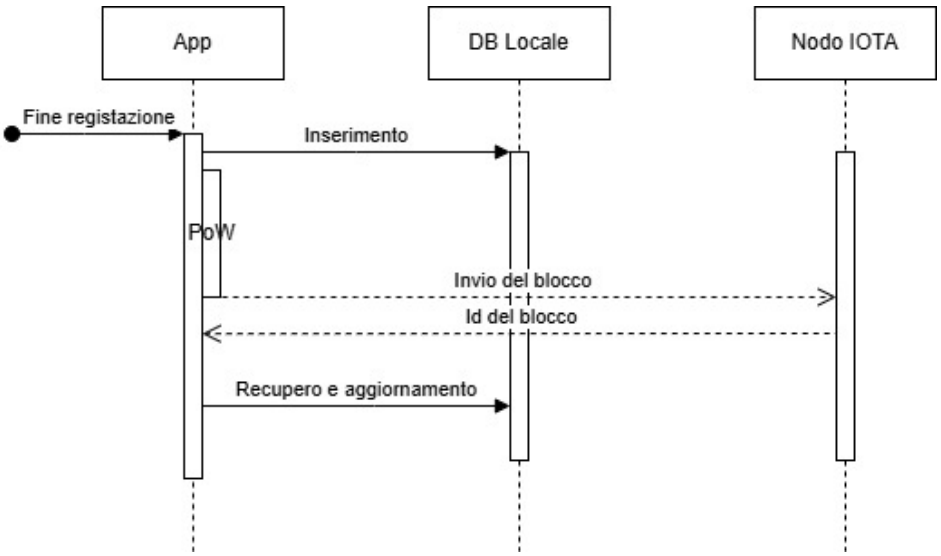

Figura 3.2: Salvataggio.

3.2 Flusso operativo

Il flusso operativo del sistema è suddiviso in due fasi principali: la raccolta dei dati e il loro invio alla DLT. Queste due componenti lavorano in sinergia per garantire la registrazione affidabile delle interazioni con Android Auto e con il dispositivo.

La raccolta dei dati viene gestita tramite un servizio in background che si avvia automaticamente all'inizio di una registrazione. Questo servizio è progettato per funzionare in modalità continua fino a quando non viene interrotto manualmente oppure nel caso della registrazione automatica, fino allo scadere dell'intervallo di tempo configurato. Quando il processo viene avviato, il sistema inizializza un oggetto di registrazione; in questa fase il sistema associa un timestamp di inizio e recupera l'identificativo univoco della sessione. Successivamente viene avviato un ciclo iterativo che raccoglie in tempo reale gli eventi generati durante l'uso del dispositivo e di Android Auto. Questi eventi sono acquisiti utilizzando l'API `UsageStatsManager`, che fornisce informazioni dettagliate sull'utilizzo delle applicazioni.

Una volta acquisiti, i dati vengono filtrati per escludere eventi di sistema irrilevanti e mantenere solo quelli associati ad attività significative per l'utente, come l'avvio di applicazioni o l'interazione con il sistema Android Auto. Dopo il filtraggio, i dati rimanenti vengono salvati in un database locale dove vengono associati alla sessione di registrazione tramite l'identificativo precedentemente recuperato 3.3.

Il passo successivo nel flusso operativo consiste nell'invio delle registrazioni completate alla DLT. Questo processo è gestito da un servizio in background dedicato. Tale servizio

può essere attivato in due modalità: manualmente, quando l'utente preme il pulsante di invio o automaticamente, allo scadere dell'intervallo configurato durante la registrazione automatica.

Le registrazioni pronte per l'invio vengono aggiunte a una coda di elaborazione, che consente di gestire in modo efficiente più richieste contemporaneamente, evitando il mancato invio. Ogni richiesta nella coda segue un flusso ben definito; il primo passo consiste nella richiesta di tips alla rete, cioè una lista di massimo 4 blocchi parents a cui si può collegare il blocco in costruzione, poi c'è la serializzazione dei dati raccolti, un processo che converte gli eventi in un formato compatto e strutturato adatto alla trasmissione. Successivamente, i dati serializzati vengono codificati per garantire integrità e compatibilità con il protocollo della DLT.

Il passo successivo è la ricerca del *nonce*, un valore numerico che soddisfa una condizione crittografica predefinita. Questa operazione, nota come *Proof of Work* (PoW), è necessaria per garantire la validità della transazione nel contesto di alcune reti DLT come IOTA. Una volta trovato il nonce corretto, i dati vengono inviati tramite una chiamata HTTP alle API di IOTA per la registrazione sul Tangle.

Il sistema include un meccanismo di gestione degli errori per garantire l'affidabilità del processo. Se la registrazione sulla DLT fallisce (ad esempio, se ai tips precedentemente richiesti vengono associati altri blocchi durante la ricerca del nonce), la richiesta viene automaticamente reinserita nella coda per essere rielaborata. Questo approccio assicura che nessun dato venga perso anche in caso di problemi temporanei. Il processo termina quando la coda delle richieste è completamente vuota 3.4.

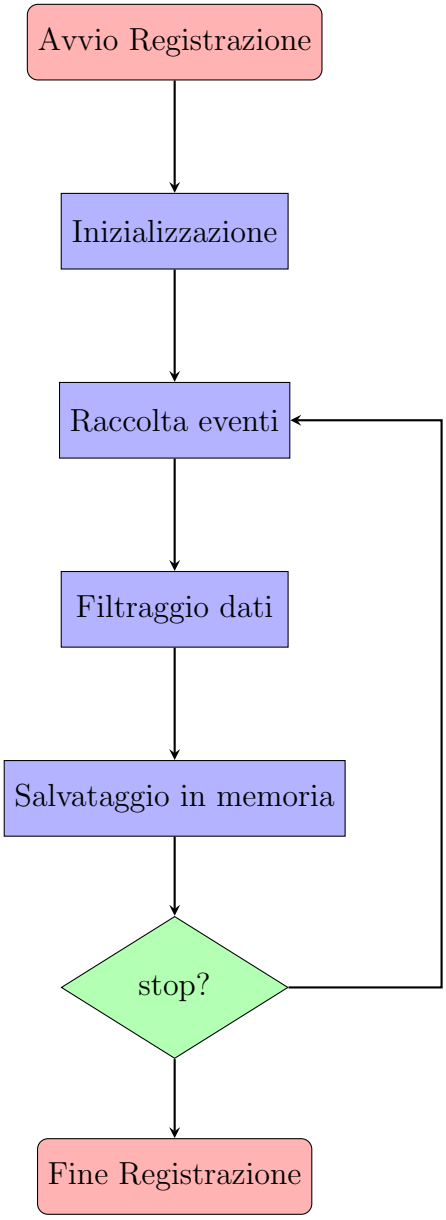

Figura 3.3: Flusso di registrazione degli eventi

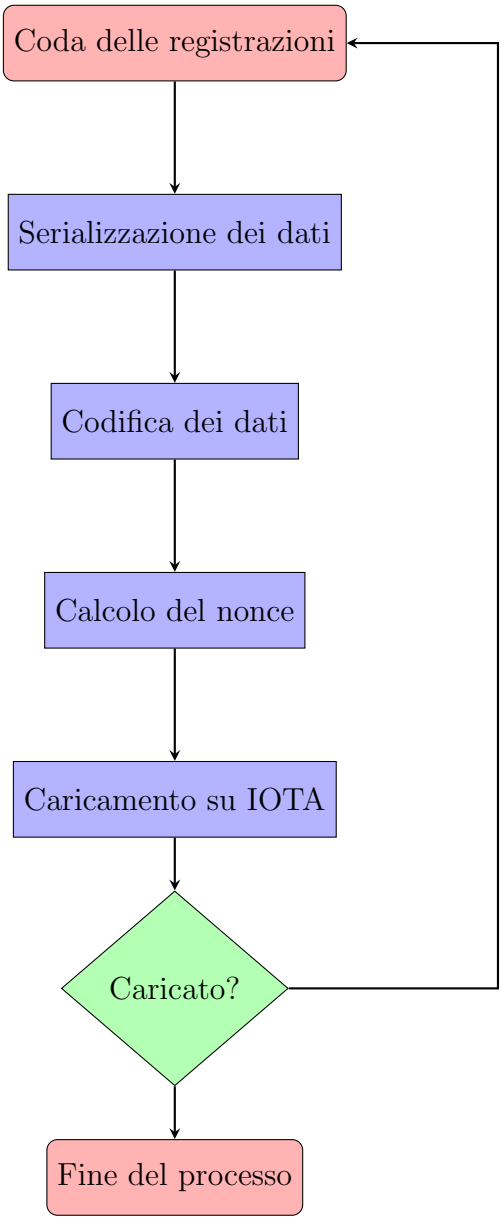

Figura 3.4: Flusso di caricamento su IOTA

Capitolo 4

Implementazione

In questo capitolo viene presentata l'implementazione dell'architettura descritta nel Capitolo 3. La progettazione ha portato allo sviluppo di un'applicazione Android denominata "Blackbox" che funge da client per la detection di attività su Android Auto e per trasmettere queste registrazioni alla DLT di IOTA.

L'applicazione implementa il modello di interazione discusso precedentemente, integrando il monitoraggio degli eventi attraverso l'API di Android, la gestione dei dati raccolti tramite un database locale e il caricamento sicuro su IOTA.

Il client si interfaccia direttamente con l'utente offrendo modalità di registrazione manuali e automatiche; inoltre incorpora notifiche informative per garantire la consapevolezza dell'utente durante le operazioni in background.

Repository Il codice dell'applicazione è consultabile nella repository `https://github.com/AlexandruNicolescu00/blackbox`

Le successive sezioni mostrano: una panoramica dell'applicazione, una descrizione delle tecnologie e librerie utilizzate e un approfondimento dei moduli più rilevanti per il funzionamento dell'app.

4.1 Panoramica dell'app

Di seguito una panoramica delle schermate principali dell'app

Home La schermata principale dell'app funge da punto di ingresso e presenta una breve introduzione sulle funzionalità chiave dell'applicazione. In questa schermata, l'utente può selezionare la modalità di registrazione: manuale o automatica.

In modalità **manuale**, il controllo è interamente lasciato all'utente, che può avviare, interrompere o inviare i dati raccolti alla DLT in base alle proprie esigenze. In modalità **automatica**, l'utente può configurare un intervallo temporale specifico, espresso in secondi, che determina ogni quanto tempo, in presenza di eventi rilevati, i dati vengono inviati automaticamente alla DLT. Questa modalità è pensata per garantire un'operatività fluida, riducendo l'interazione necessaria con l'app durante la guida.

Un'interfaccia minimalista che consente di accedere rapidamente alle configurazioni essenziali. La modalità automatica, in particolare, è progettata per minimizzare le distrazioni, una caratteristica fondamentale per la sicurezza.

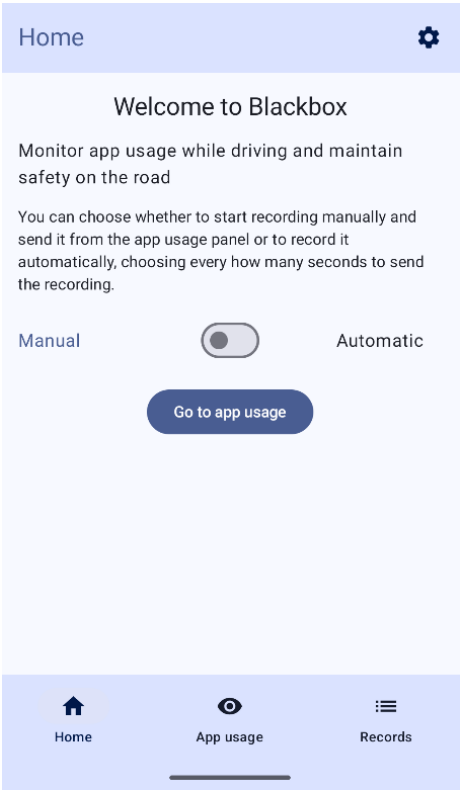

Figura 4.1: Home manual.

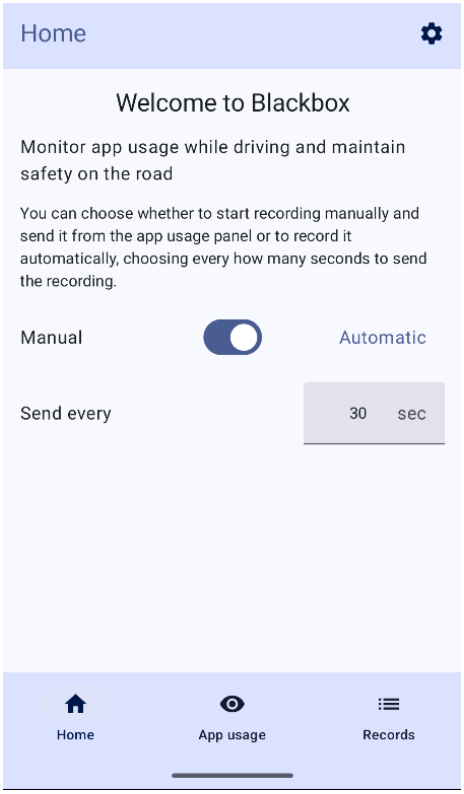

Figura 4.2: Home auto.

App usage In questa schermata è concentrata l'operatività dell'applicazione, dove vengono mostrati in tempo reale gli eventi rilevati relativi all'utilizzo del dispositivo e di Android Auto. Per le registrazioni manuali, l'interfaccia fornisce pulsanti dedicati per avviare o interrompere la raccolta dei dati e un pulsante per caricare manualmente i dati sulla DLT. Al contrario, nella modalità automatica, queste operazioni vengono eseguite automaticamente seguendo l'intervallo temporale configurato nella schermata principale.

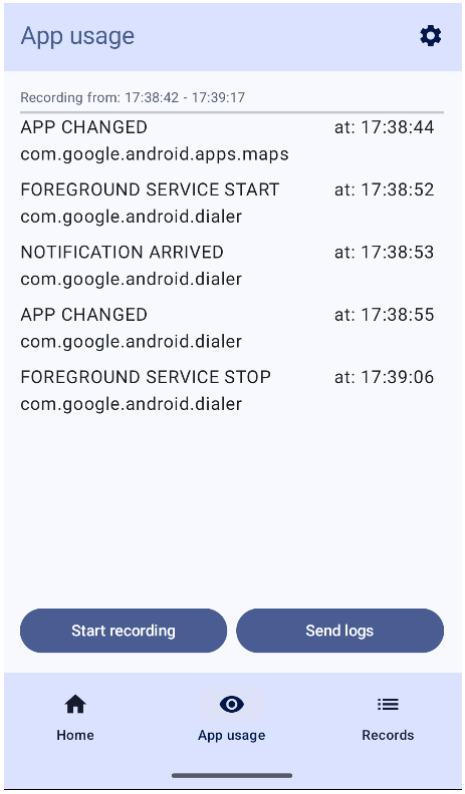

Figura 4.3: App usage manual.

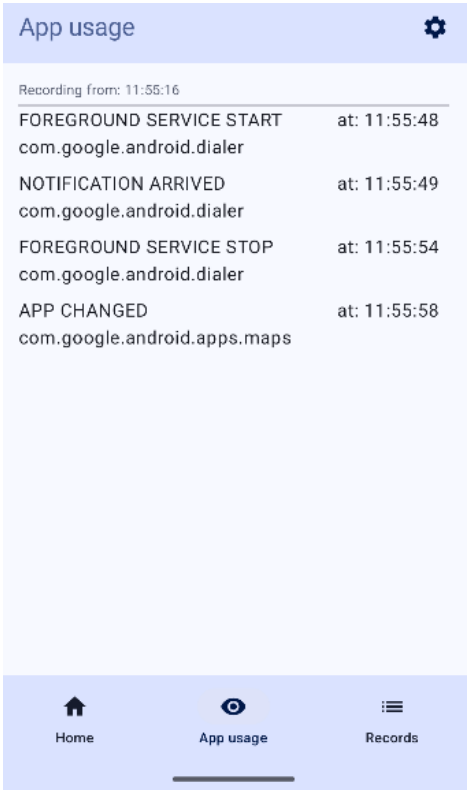

Figura 4.4: App usage auto.

Quando viene avviata una registrazione, una notifica informativa viene inviata al dispositivo per segnalare che il processo è in corso. Questa notifica risulta particolarmente utile nel caso di registrazioni in background, assicurando che l'utente sia consapevole dell'operazione in corso senza dover interagire direttamente con l'applicazione.

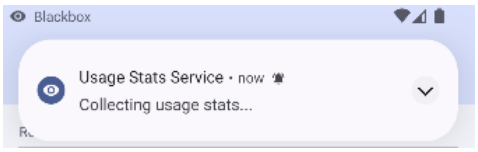

Figura 4.5: Records list.

Analogamente, durante il caricamento dei dati sulla DLT, un'altra notifica informa l'utente del caricamento in atto. Al completamento dell'operazione, la notifica scompare, i dati sono stati caricati con successo ed è possibile visualizzarli utilizzando l'explorer della rete IOTA.

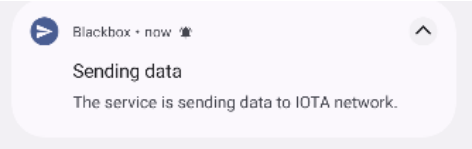

Figura 4.6: Records list.

Records La schermata *Records* offre una panoramica organizzata delle registrazioni effettuate. Le registrazioni sono raggruppate per data e possono essere ordinate in modo ascendente o discendente per facilitare la ricerca di una specifica sessione.

Per migliorare la leggibilità, le registrazioni già inviate sulla DLT sono evidenziate con un colore distintivo e accompagnate da un'icona specifica che ne facilita l'identificazione. Questo consente all'utente di distinguere rapidamente tra registrazioni locali e quelle già archiviate in modo sicuro sulla DLT.

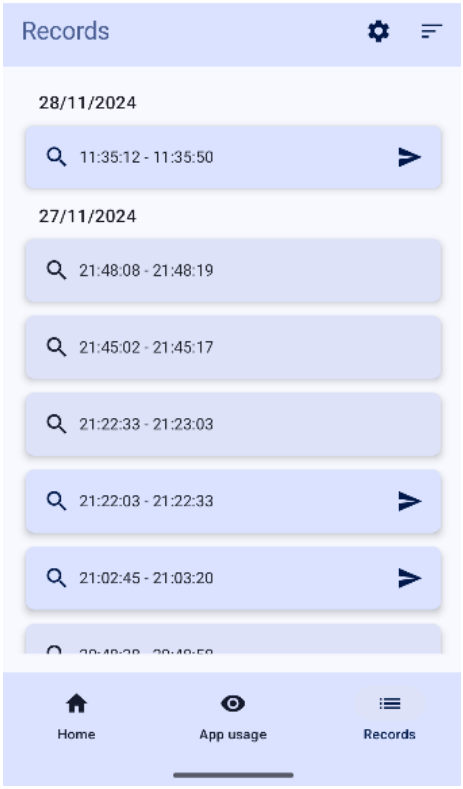

Figura 4.7: Records list.

Record detail Accedendo a una registrazione specifica, l'utente viene reindirizzato alla schermata *Record Detail* che fornisce una visione approfondita degli eventi registrati

durante quella sessione. Qui vengono elencati tutti gli eventi raccolti con descrizione e il momento della rilevazione.

Nel caso in cui la registrazione sia stata caricata sulla DLT, vengono visualizzati ulteriori dettagli, inclusi l'identificativo univoco del blocco e un pulsante che consente di aprire l'explorer della rete. Questo permette all'utente di verificare autonomamente l'integrità e la presenza dei dati sulla DLT, garantendo trasparenza e tracciabilità.

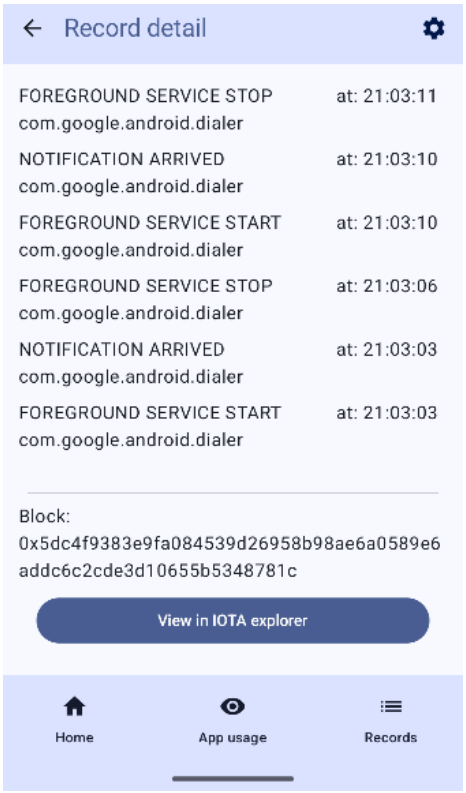

Figura 4.8: Record detail.

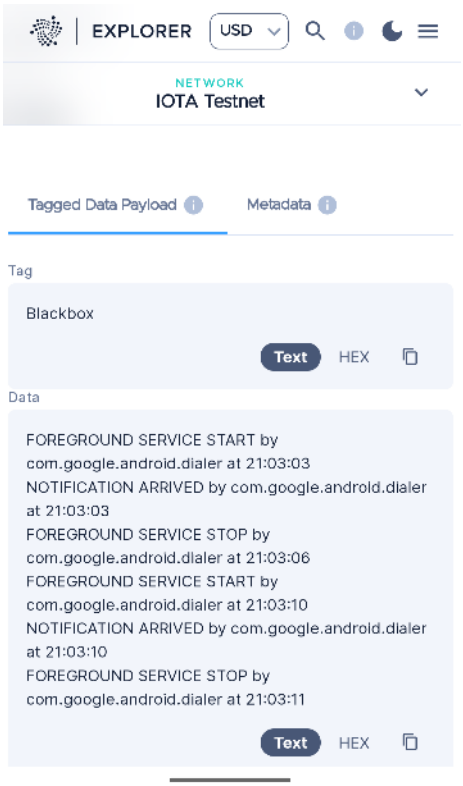

Figura 4.9: IOTA explorer.

Aspetti di design e notifiche Il design dell'app è stato pensato per minimizzare le distrazioni durante la guida e per massimizzare l'efficienza operativa. Le notifiche hanno il ruolo di tenere l'utente aggiornato senza interrompere l'esperienza di guida. Inoltre, ogni schermata è stata pensata per garantire un'interazione fluida, con elementi ben visibili e facili da selezionare.

Flusso di utilizzo L'utente inizia configurando le modalità di registrazione nella schermata *Home*, passa poi a monitorare gli eventi nella schermata *App Usage*, consulta le registrazioni passate in *Records* e visualizza i dettagli delle sessioni archiviate nella schermata *Record Detail*.

4.2 Tecnologie utilizzate

L'applicazione è stata sviluppata utilizzando diverse tecnologie e librerie per garantire la funzionalità richiesta. Di seguito sono descritte le principali tecnologie impiegate.

Android e Kotlin L'app è sviluppata su Android utilizzando Kotlin come linguaggio principale. Kotlin offre una sintassi meno verbosa rispetto a Java ed è il linguaggio ufficiale per lo sviluppo di applicazioni native Android; inoltre supporta le coroutine, utili per gestire operazioni asincrone come il caricamento di dati su IOTA. La scelta di Android permette di sfruttare il supporto nativo per le API richieste come UsageStatsManager.

Jetpack Compose L'interfaccia utente è costruita interamente con Jetpack Compose, il framework dichiarativo di Android. Questo consente di definire le UI direttamente nel codice, eliminando la necessità di file XML. Compose permette una gestione dello stato ottimizzata e l'integrazione con altre componenti Jetpack, come i ViewModel. La libreria grafica Material3 viene utilizzata per rispettare le linee guida di design moderne.

Room Per la gestione dei dati locali è utilizzata Room, una libreria per l'accesso a database SQLite. Room offre un'API basata su annotazioni per definire tabelle, query e operazioni CRUD. Inoltre, è integrata con coroutine per gestire i dati in modo asincrono. Questa libreria è impiegata per memorizzare eventi raccolti e registrazioni.

UsageStatsManager API La raccolta degli eventi del dispositivo e di Android Auto è implementata utilizzando l'API UsageStatsManager (`https://developer.android.com/reference/android/app/usage/UsageStatsManager`), nativa di Android. Questa API consente di accedere a statistiche di utilizzo delle applicazioni e di monitorare eventi specifici come l'avvio o la chiusura di app. L'API richiede permessi avanzati per accedere ai dati che vengono gestiti nel flusso di autorizzazione dell'app.

Retrofit L'invio delle registrazioni sulla DLT è implementato tramite Retrofit, una libreria per effettuare chiamate HTTP. Gli endpoint REST sono definiti in interfacce, mentre Gson viene utilizzato per serializzare e deserializzare i dati JSON. Retrofit combinato con OkHttp garantisce la gestione efficiente delle connessioni di rete.

CryptoHash La libreria CryptoHash è utilizzata nel processo di Proof-of-Work richiesto dal protocollo IOTA. In particolare, è impiegata per calcolare l'hash BLAKE2b-256 del messaggio serializzato, escludendo il campo Nonce di 8 byte. Questo passaggio rappresenta la fase iniziale della validazione crittografica necessaria per trovare un nonce valido e completare il processo di registrazione su IOTA.

IOTA IOTA è il sistema di registro distribuito utilizzato per archiviare i dati delle registrazioni. La scelta di IOTA è stata guidata principalmente dall'assenza di costi di transazione e dalla velocità di elaborazione, aspetti cruciali per un sistema progettato per gestire un elevato numero di registrazioni.

L'app comunica con IOTA direttamente tramite le API (`https://wiki.iota.org/apis/welcome/`), poiché al momento non esiste un SDK IOTA per Android. L'adozione di un'applicazione backend separata, scritta in uno dei linguaggi supportati, è stata esclusa per poter garantire l'integrità delle registrazioni.

4.3 Moduli

Il sistema sviluppato per il monitoraggio degli eventi e la registrazione dei dati sulla DLT di IOTA si basa su un'architettura modulare per garantire efficienza e manutenibilità. I componenti principali dell'implementazione sono il sistema di monitoraggio degli eventi e il meccanismo di caricamento dei dati.

4.3.1 Monitoraggio degli eventi

UsageStatsService Il monitoraggio degli eventi è gestito da un servizio denominato `UsageStatsService`, progettato come un *foreground service* per assicurare l'esecuzione continua anche quando l'applicazione è in background. Questo approccio evita che il sistema termini il servizio per liberare risorse, garantendo un flusso costante di dati.

L'integrazione dell'API `UsageStatsManager` tramite gli eventi forniti da `UsageEvents` ha permesso di ottenere una raccolta dettagliata di informazioni. Tuttavia è emersa una significativa limitazione poiché Android non offre API native progettate per monitorare l'interazione all'interno delle applicazioni. Di conseguenza è stato necessario adottare un sistema di polling che interrogasse l'API ogni secondo su quali eventi sono accaduti. Questo approccio però presenta dei limiti nel livello di dettaglio delle rilevazioni, è possibile rilevare cambi di app, notifiche e foreground service quali telefonate, però non permette di rilevare l'utilizzo in app come scrivere un messaggio o cercare una canzone che sono attività sicuramente pericolose durante la guida. Nonostante questi vincoli, il compromesso ottenuto rappresenta un miglioramento rispetto all'implementazione iniziale basata `UsageStats` (`https://developer.android.com/reference/android/app/usage/UsageStats`) che limitava la raccolta di dati al solo ultimo accesso registrato di un'app.

`UsageStatsService` monitora gli eventi di utilizzo come i cambi di applicazione o le notifiche ricevute e fornisce i dati della registrazione al resto dell'applicazione, supporta due modalità di registrazione: manuale e automatica. Quando il servizio viene avviato esegue alcune operazioni iniziali tra cui la creazione di una notifica per informare l'utente

che il monitoraggio è attivo e l'avvio di coroutine per gestire l'intervallo di raccolta dati configurato dall'utente e la modalità di registrazione scelta. In caso di modifica della modalità da automatica a manuale, il servizio si adegua interrompendo la registrazione in corso.

La raccolta automatica è gestita dalla funzione `startAutoCollection`, che avvia un ciclo per recuperare gli eventi in base a intervalli di tempo predefiniti, mostrando i dati quasi in tempo reale grazie a un intervallo di aggiornamento impostato a un secondo. La modalità manuale invece è gestita dalla funzione `startUsageEventsCollection` che esegue il recupero continuo degli eventi fino a che non viene esplicitamente interrotta. Il cuore del sistema di monitoraggio è rappresentato dalla funzione `requestEvents`, che interroga la classe `usageStatsManager` gli eventi avvenuti dall'ultima richiesta, poi applica un filtro per escludere gli eventi non rilevanti e, in caso di mancati permessi, ferma la registrazione. Successivamente, per ogni evento rimasto, lo categorizza in base alla tipologia (activity resumed, foreground service start, foreground service stop, notification [1]), crea un oggetto di tipo `UsageEvent` da memorizzare nel database e aggiorna lo stato della registrazione, cioè una variabile condivisa che permette di visualizzare i dati sull'interfaccia utente.

AppUsageStatsManager Questa classe fa uso dell'API `UsageStatsManager` che consente di accedere alle statistiche di utilizzo del dispositivo. Per utilizzare questa API è necessario il permesso `android.permission.PACKAGE_USAGE_STATS` che richiede l'autorizzazione esplicita dell'utente.

La funzione `getUserEvents` utilizza l'API per recuperare gli eventi di utilizzo tra un intervallo temporale; la funzione per prima cosa controlla se l'app ha i permessi per poi inizializzare il servizio di sistema `USAGE_STATS_SERVICE`, quindi con il servizio chiama la funzione `queryEvents` per recuperare un flusso di eventi, che poi viene elaborato e restituito al chiamante della funzione.

```
fun getUserEvents(beginTime: Long, endTime: Long)
    : List<UsageEvents.Event> {
    if (!permissionsState.value.hasUsageStatsPermission) {
        throw SecurityException("Usage stats permission not granted
            ${permissionsState.value.hasUsageStatsPermission}")
    }
    if (usageStatsManager == null) {
        usageStatsManager = getUsageStatsManager()
    }
    val usageEvents = usageStatsManager!!.queryEvents(beginTime, endTime)
    val events = mutableListOf<UsageEvents.Event>()

    if (usageEvents != null) {
        while (usageEvents.hasNextEvent()) {
```

[1] Il `NOTIFICATION_EVENT` è una costante con valore 10 che dai test effettuati risulta indicare l'arrivo di una notifica, ma questo non è riportato nella documentazione (`https://developer.android.com/reference/android/app/usage/UsageEvents.Event`)

```
15              val eventAux = UsageEvents.Event()
16              events.add(eventAux)
17              usageEvents.getNextEvent(eventAux)
18          }
19      }
20
21      return events
22  }
```

4.3.2 Caricamento su IOTA

Per il caricamento dei dati sulla DLT, il sistema utilizza un approccio modulare suddiviso in gestione della coda, preparazione della richiesta e il processo di Proof of Work (PoW).

La mancanza di un SDK ufficiale per Android in grado di interagire con IOTA ha rappresentato una delle maggiori sfide affrontate durante lo sviluppo. Questa mancanza ha richiesto l'implementazione manuale di funzionalità chiave per il progetto, quali l'interazione con le API di IOTA e il processo di Proof of Work, aumentando notevolmente i tempi di sviluppo, in particolare perchè gli algoritmi `Curl-P-81` e `b1t6` non hanno delle documentazioni tecniche e quindi è stato possibile implementarli solamente consultando la loro implementazione sugli SDK ufficiali. Questo però ha permesso di vedere in maniera approfondita il meccanismo di validazione dei blocchi di IOTA. L'utilizzo di un backend separato per usare l'SDK avrebbe sicuramente accorciato i tempi di sviluppo, ma avrebbe introdotto problemi di integrità e ritardi nella trasmissione dei dati compromettendo l'affidabilità del sistema come discusso nel terzo capitolo.

Gestione della coda Il servizio `SendDataService` si occupa della gestione della coda. Quando arriva una richiesta di invio, questa viene accodata dalla funzione `enqueueRequest` per essere elaborata dalla funzione `processQueue`, la quale per ogni elemento della coda inoltra la richiesta alla funzione `sendData` e attende la sua terminazione, che avviene quando il caso d'uso `SendData` ritorna un successo o un errore. In caso di errori, ad esempio la perdita di connessione, le richieste vengono reinserite nella coda per poter essere rielaborate, garantendo una maggiore resilienza del sistema.

Preparazione e invio della richiesta La preparazione e l'invio della richiesta è svolta dal caso d'uso `SendData` che inizialmente prepara il blocco da inviare secondo le specifiche (`https://wiki.iota.org/apis/core/v2/submit-a-block/`), durante la creazione del blocco si richiede una lista di parents alla rete, ai quali si può agganciare il nuovo blocco; successivamente delega alla classe `PoW` la ricerca del nonce per poter completare il blocco e inviarlo. In questo processo, per garantire la sua integrità, vengono gestiti errori di mancata connessione durante l'invio ed errori di comunicazione con il nodo, che potrebbero avvenire se ai parents inizialmente richiesti sono stati concatenati altri blocchi.

Ricerca del nonce La ricerca del nonce è una componente fondamentale per il funzionamento del sistema, implementato dalla funzione `performPow`. Questa ricerca segue il protocollo Proof of Work richiesto da IOTA come descritto nella documentazione ufficiale (`https://wiki.iota.org/tips/tips/TIP-0012/`):

1. Calcola l'hash Blake2b-256 del messaggio serializzato, escludendo il nonce
2. Converte l'hash nella rappresentazione a 192 trit codificato tramite l'algoritmo b1t6 (1 byte 6 trit), mentre un bit può avere valori 0 o 1, un trit può avere valori -1, 0 e 1.
3. Prende il nonce nella sua rappresentazione a 8 byte in little endian e lo converte nella rappresentazione a 48 trit, codificato tramite l'algoritmo b1t6.
4. Prepara l'input per l'algoritmo di hash Curl-P-81 agganciando l'hash del messaggio al nonce e aggiunge tre zeri di padding per arrivare ad avere un hash di lunghezza 243 (192 + 48 + 3), cioè la lunghezza richiesta da Curl-P-81.
5. Calcola l'hash tramite l'algoritmo Curl-P-81.
6. Conta il numero di zero finali dell'hash precedentemente calcolato.
7. Calcola il valore di Proof of Work con la formula 3^{zeri}/size(message): se il valore calcolato è maggiore o uguale alla costastante `MIN_POW_SCORE`, ovvero il valore minimo accettato dalla rete, allora il nonce trovato valida il messaggio.

```
override fun performPow(block: FullBlockWithTaggedDataPayload): ULong {
    val serializedMessage = serializeMessage(block)
    val powDigest = blake2b(serializedMessage)
    val powDigestTrits = b1t6Encode(powDigest)
    var nonce = 0uL
    val curl = JCurl()
    while (true) {
        val nonceBites = nonce.toLittleEndianByteArray(8)
        val nonceTrits = b1t6Encode(nonceBites)
        val inputTrits = powDigestTrits + nonceTrits + byteArrayOf(0, 0, 0)
        curl.absorb(inputTrits)
        val powHash = ByteArray(HASH_LENGTH)
        curl.squeeze(powHash)
        val trailingZeros = countTrailingZeros(powHash)
        val message = serializedMessage + nonceBites
        val score = Math.pow(3.0, trailingZeros.toDouble()) / message.size
        if (score >= MIN_POW_SCORE) {
            Log.d("Nonce", "Score: $score")
            return nonce
        }
        nonce++
        curl.reset()
    }
}
```

La funzione `serializeMessage` serializza il messaggio rispettando la struttura indicata nella documentazione (`https://wiki.iota.org/tips/tips/TIP-0024/`)

L'hash Blake2b-256 è calcolato tramite la libreria `cryptohash`. BLAKE2b (`https://datatracker.ietf.org/doc/html/rfc7693`) è una funzione di hash progettata per essere più veloce e sicura di MD5 e SHA-256.

Come riportato nella documentazione ufficiale (`https://wiki.iota.org/tips/tips/TIP-0005/`) la codifica b1t6 è un metodo utilizzato per convertire dati binari in una rappresentazione ternaria. Questo processo è necessario perchè nel protocollo IOTA una transazione è rappresentata come dato ternario, ma deve comunque supportare dati binari per gli hash. L'implementazione ha seguito l'esempio della codifica scritta in `go` (`https://github.com/Wollac/iota-crypto-demo/blob/master/pkg/encoding/b1t6/b1t6.go`) riportata nella documentazione ufficiale, adattandolo per il linguaggio `Kotlin`, codice riportato in appendice 1.

Curl-P-81 è una funzione di hash ternaria utilizzata nel protocollo IOTA per il calcolo del Proof of Work. La classe `JCurl` è stata implementata seguendo l'implementazione dell'SDK Java deprecato (`https://github.com/iotaledger/iota-java/blob/dev/jota/src/main/java/org/iota/jota/pow/JCurl.java`) e adattata al linguaggio `Kotlin`, il codice è riportato in appendice 2. Questo è stato possibile poiché la logica di Curl-P-81 è rimasta invariata nell'attuale SDK (`https://github.com/iotaledger/iota.go/blob/legacy/curl/curl.go`)

La funzione accetta una stringa di 243 trits come input, si basa su tre operazioni:

- **absorb** copia l'input nello stato interno e chiama `transform` per modificare lo stato interno
- **squeeze** copia lo stato interno nell'output e chiama `transform` per modificare lo stato interno
- **transform** esegue per 81 volte le seguenti operazioni:
  - copia lo stato interno in un buffer temporaneo
  - ogni valore dello stato interno viene modificato con un valore della tabella di verità in base a un indice dinamicamente calcolato

Capitolo 5

Valutazione e risultati

La valutazione del sistema sviluppato è un passaggio fondamentale per mostrare l'efficacia, le prestazioni e i limiti della soluzione proposta. Questo capitolo presenta i risultati delle sperimentazioni condotte per analizzare il comportamento dell'applicazione in vari scenari, valutandone sia gli aspetti tecnici che l'impatto sull'efficienza del sistema complessivo.

Gli esperimenti si concentrano principalmente su tre aree: il costo delle operazioni sulla DLT, il comportamento del Proof of Work in relazione ai dati da memorizzare e l'effetto del carico della rete sulla velocità di caricamento dei blocchi.

L'obiettivo è quello di verificare che il sistema rispondesse in maniera affidabile alle esigenze per cui è stato progettato. Per fare ciò, sono stati progettati test mirati a simulare scenari d'uso realistici, ad esempio l'invio di blocchi aventi un numero di eventi crescenti o l'interazione con la DLT durante periodi di congestione della rete. Gli esperimenti sono stati analizzati per evidenziare potenzialità e limitazioni del sistema, fornendo indicazioni per sviluppi futuri e ottimizzazioni.

5.1 Test di evaluation

In questa sezione verranno discussi i test svolti per analizzare le prestazioni e i costi del sistema sviluppato.

Gas fee L'applicazione sviluppata fa uso del protocollo Stardust presente nell'attuale mainnet di IOTA, che consente l'invio di `Tagged Data Block` senza alcun costo. Questo tipo di blocco è progettato per memorizzare dati arbitrari senza richiedere gas fee o vincoli di bilancio. Di conseguenza, l'utilizzo dell'applicazione risulta completamente gratuito dal punto di vista delle transazioni sulla DLT.

Calcolo del nonce in base al numero di eventi Per analizzare l'impatto del numero di eventi sul calcolo del nonce durante il processo di Proof of Work, è stato condotto un test simulato. In questo esperimento sono stati generati blocchi aventi tutti i campi costanti tranne i dati del payload. I dati del payload avevano un numero crescente di eventi, da 0 a 100. Gli eventi sono stati selezionati casualmente da un campione predefinito di eventi più comuni e concatenati in una stringa.

```
val events = buildString {
                    repeat(numberOfEvents) {
                        val random = Random.nextInt(sample.size)
                        append(sample[random]).append("\n")
                    }
                }
var block = FullBlockWithTaggedDataPayload(
                protocolVersion = 2,
                parents = repository.getTips().tips,
                payload = Payload(
                    type = 5,
                    tag = BLOCK_TAG,
                    data = events
                ),
                nonce = ""
            )
```

Per valutare i risultati si è deciso di utilizzare il valore del nonce e non il tempo di calcolo in quanto quest'ultimo dipende principalmente dal dispositivo sul quale viene eseguito, mentre il nonce rappresenta un numero fisso di operazioni eseguite. I risultati sono rappresentati in questo grafico 5.1

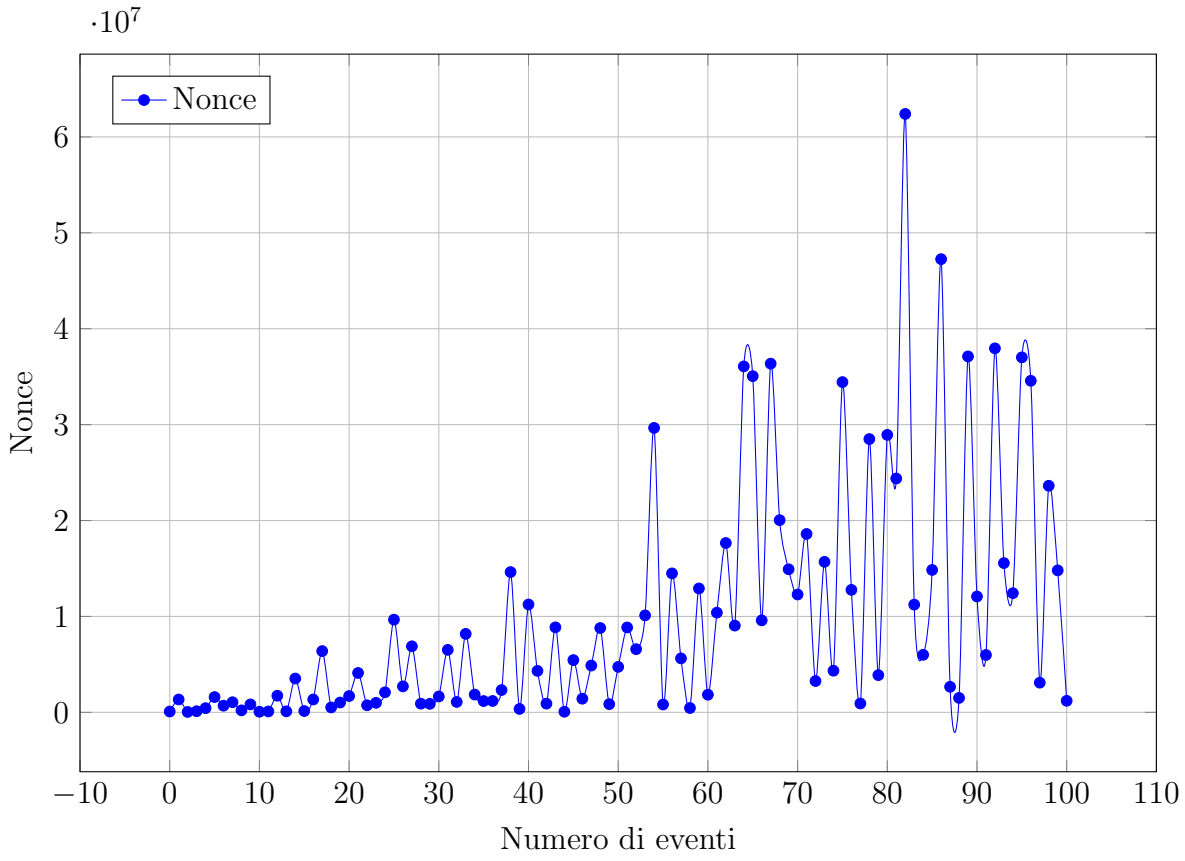

Figura 5.1: Relazione tra il numero di eventi e il valore del nonce.

Da questo possiamo trarre le seguenti considerazioni:

1. Il valore del nonce non cresce linearmente con l'aumento del numero di eventi
2. Nonostante la variabilità casuale, i risultati mostrano una tendenza generale all'aumento del nonce con l'aumentare del numero degli eventi.
3. Alcuni valori del nonce mostrano oscillazioni imprevedibili anche per payload di lunghezza simile e queste oscillazioni sono più frequenti con l'aumentare del numero di eventi

Valutazione del ritardo dovuto al flooding della rete L'esperimento ha lo scopo di valutare il ritardo introdotto dall'invio di un blocco quando la rete è sotto carico, registrando un numero elevato di blocchi al secondo. Durante la fase di test, svolta su un dispositivo emulato avente una cpu a 8 core e 6144 MB di memoria ram sulla rete IOTA Testnet che registrava una media di 7 blocchi al secondo, il dato è stato visualizzato dall'explorer (`https://explorer.iota.org/iota-testnet`). Ogni blocco

è stato generato con un payload contenente eventi casuali, simulando scenari realistici di utilizzo. Durante l'esperimento sono stati utilizzati cinque thread paralleli per gestire l'invio simultaneo di 100 blocchi.

I risultati sono mostrati nel grafico 5.2 dove viene riportato in blu il primo nonce calcolato e in rosso la somma dei nonce calcolati al momento dell'effettivo invio sulla DLT.

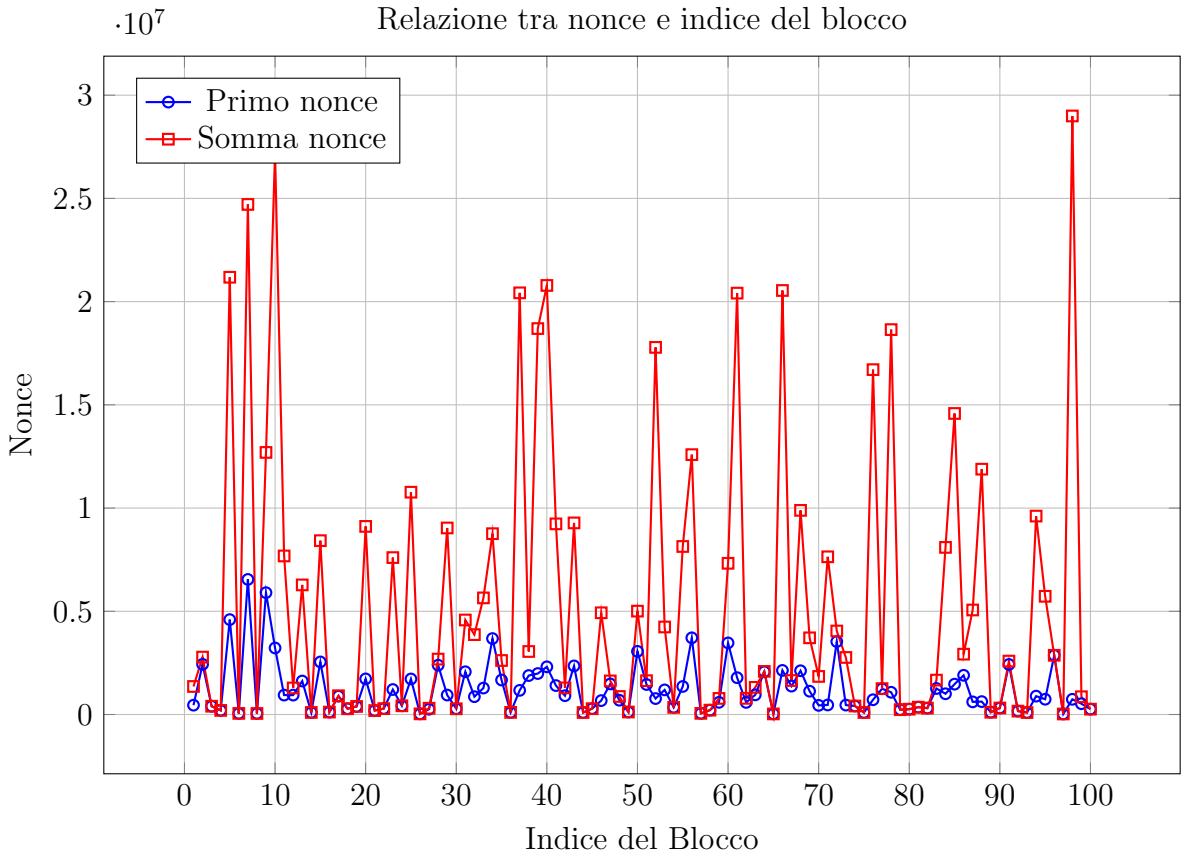

Figura 5.2: Relazione tra nonce e indice del blocco in due metriche

Visto l'alto numero di tentativi effettuati da alcuni blocchi, è stato condotto un ulteriore esperimento utilizzando una variante della funzione `performPow`, chiamata `performParallelPow`. Questa variante implementa il processo di ricerca del nonce in parallelo sfruttando 6 corourine per velocizzare il processo.

Il numero di coroutine è stato selezionato mediante un processo di tuning volto a massimizzare il numero di nonce verificati al secondo sul dispositivo emulato. Si è calcolata una media sul tempo di calcolo di 10 nonce su blocchi generati casualmente, i risultati sono stati i seguenti: con 5 coroutine la media è stata di 26082,3 nonce verificati al secondo, con 6 coroutine la media è stata di 29988,6 nonce verificati al secondo, da 7 coroutine

in poi la media risultava minore in quanto la memoria raggiungeva velocemente il limite facendo attivare spesso il garbage collector, andando a rallentare il calcolo. Invece il test precedente 5.2 è stato svolto con una media di 5146,5 nonce verificati al secodo.

L'obiettivo era ridurre notevolmente il tempo trascorso tra la richiesta dei parents e dell'invio del blocco. Il codice della funzione è riportato in appendice 3. Questi sono stati i risultati.

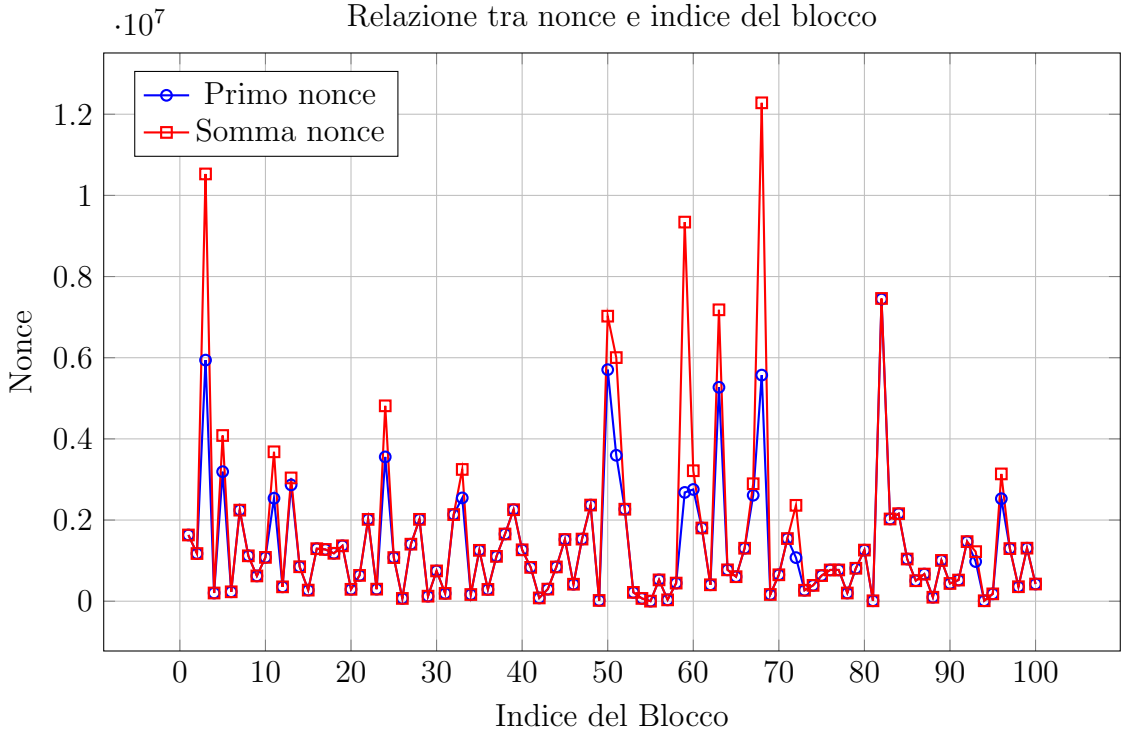

Figura 5.3: Relazione tra nonce e indice del blocco per 100 blocchi

In questo secondo grafico 5.3 troviamo uno scenario molto differente, molti blocchi sono stati caricati al primo tentativo e solo alcuni blocchi hanno impiegato tentativi multipli.

5.2 Limitazioni del sistema

Un'importante limitazione emersa durante lo sviluppo riguarda il comportamento di Android Auto nel gestire le applicazioni visualizzate sullo schermo. Android Auto è un intermediario tra le app del dispositivo e il sistema del veicolo e non espone il `packageName` delle applicazioni di terze parti, rendendo impossibile identificarle tramite le API di Android. A tutte le applicazioni viene associato il nome del pacchetto di Android Auto, ovvero `com.google.android.projection.gearhead`. Tuttavia,

alcune applicazioni sviluppate da Google, come Google Maps e Google Assistant, sono riconoscibili grazie ai rispettivi `packageName`, `com.google.android.apps.maps` e `com.google.android.googlequicksearchbox`. Questo comportamento limita la capacità di monitorare con precisione quali applicazioni sono effettivamente utilizzate durante l'utilizzo di Android Auto.

Un'altra limitazione del sistema riguarda i tempi richiesti per il caricamento dei dati sulla DLT di IOTA. Questo processo non è istantaneo in quanto include il processo del Proof of Work e la trasmissione della transazione al nodo della rete. Il tempo totale può variare da pochi secondi a periodi più lunghi. In situazioni come incidenti stradali, dove è necessaria una comunicazione tempestiva per evitare il mancato invio dovuto a danni al dispositivo, questo ritardo potrebbe non essere ideale.

Un'ulteriore limitazione rilevante è rappresentata dalla volatilità dell'ecosistema IOTA. Dopo lo sviluppo dell'applicazione è stato annunciato il completo rimpiazzo del protocollo **Stardust** con il nuovo protocollo **Move-based ledger** descritto nella nuova documentazione di IOTA (`https://docs.iota.org/developer/stardust/stardust-migration`). La transizione implica l'adozione di nuove API perchè il nuovo protocollo sostituisce le API attualmente in uso e introduce il gas fee per il calcolo della transazione e per l'utilizzo di storage.

Capitolo 6

Conclusioni e sviluppi futuri

In questo progetto di tesi abbiamo presentato un'applicazione per la detection dell'utilizzo di Android Auto durante la guida, affrontando le sfide legate alla raccolta in tempo reale dei dati di utilizzo e all'integrazione del sistema con la DLT di IOTA.

Dopo aver introdotto il contesto tecnologico e motivato la necessità di uno strumento di questo tipo, lo studio ha approfondito il panorama dello stato dell'arte. Sono stati analizzati i rischi legati alle distrazioni del conducente e le tecnologie già utilizzate per rilevare comportamenti rischiosi. Inoltre, sono state esplorate le caratteristiche di DLT come IOTA, con il loro potenziale per garantire la trasparenza e l'immutabilità dei dati e infine l'utilizzo delle DLT in ambiti come la mobilità e la sicurezza stradale.

La progettazione del sistema ha definito un'architettura modulare che integra un'app Android con la DLT di IOTA, mostrando come queste componenti interagiscono tra loro. Le scelte progettuali sono state orientate alla scalabilità e alla robustezza. Inoltre, sono stati definiti i flussi operativi per la raccolta, il salvataggio e la trasmissione dei dati alla DLT.

L'implementazione ha visto la realizzazione dell'applicazione "Blackbox" e la selezione di tecnologie adeguate per il monitoraggio degli eventi e il caricamento dei dati sulla rete. Sono stati risolti problemi legati alla mancanza di strumenti nativi per la detection di alcune attività su Android Auto e all'assenza di un SDK ufficiale per IOTA. Questo ha richiesto lo sviluppo di soluzioni personalizzate, come l'implementazione di algoritmi necessari ad eseguire il processo di PoW.

L'analisi delle prestazioni ha evidenziato punti di forza e limiti del sistema. L'adozione del protocollo Stardust di IOTA ha garantito l'assenza di costi di transazione, rendendo il sistema economicamente sostenibile. I test hanno mostrato che il numero di eventi da registrare non è linearmente correlato al valore del nonce necessario per il PoW, anche

se si osserva una tendenza generale all'aumento della complessità con l'aumentare del numero di eventi. Tuttavia, in condizioni di congestione della rete, i tempi di registrazione si sono allungati a causa della competizione nell'invio dei blocchi. L'adozione di un approccio parallelo per il calcolo del PoW ha contribuito a ridurre significativamente questo impatto, aumentando la velocità complessiva del sistema. Infine, si sono evidenziati i limiti di tale sistema, fornendo spunti per ulteriori miglioramenti e adattamenti futuri.

Le soluzioni adottate hanno mostrato la fattibilità dell'obiettivo nello scenario proposto, evidenziando però alcune limitazioni legate all'ecosistema Android e alle peculiarità del protocollo IOTA. Tra queste, il comportamento di Android Auto che nasconde i dettagli delle applicazioni di terze parti e il ritardo introdotto dal processo di Proof of Work per la registrazione dei blocchi.

Per quanto riguarda gli sviluppi futuri, ci sono opportunità nella gestione delle registrazioni in tempo reale, nell'ottimizzazione delle prestazioni del PoW e nell'adattamento del sistema ai cambiamenti previsti per il protocollo IOTA. Ulteriori estensioni potrebbero includere l'utilizzo di strumenti alternativi per la detection di attività come le API di accessibilità e anche il riconoscimento automatico dell'inizio della guida da parte dell'utente per avviare la registrazione.

In sintesi, il progetto rappresenta un primo passo verso l'uso del dispositivo come strumento unico di monitoraggio e un'alternativa per l'utilizzo della DLT in scenari di sicurezza stradale, aprendo la strada a ulteriori innovazioni nel settore.

Bibliografia

[1] Rafael A. Berri, Alexandre G. Silva, Rafael S. Parpinelli, Elaine Girardi, and Rangel Arthur. A pattern recognition system for detecting use of mobile phones while driving. In *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*, volume 2, pages 411–418, 2014.

[2] Christian Braunagel, Enkelejda Kasneci, Wolfgang Stolzmann, and Wolfgang Rosenstiel. Driver-activity recognition in the context of conditionally autonomous driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1652–1657, 2015.

[3] Alberto Butera, Noemi Romani, Valentina Gatteschi, et al. Improving driving behavior: a blockchain-based gamification system. In *Proceedings of DLT 2024*. CEUR, 2024.

[4] Nabil El Ioini and Claus Pahl. A review of distributed ledger technologies. In Hervé Panetto, Christophe Debruyne, Henderik A. Proper, Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, pages 277–288, Cham, 2018. Springer International Publishing.

[5] Yuanzhao Fan, Fei Gu, Jin Wang, Jianping Wang, Kejie Lu, and Jianwei Niu. Safedriving: An effective abnormal driving behavior detection system based on emg signals. *IEEE Internet of Things Journal*, 9(14):12338–12350, 2022.

[6] Luc Gerrits, Thomas Mabrut, and François Verdier. Communicate vehicle accident data to blockchain for secure and reliable record-keeping using android automotive application. In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pages 22–27, 2023.

[7] Google. Android auto. `https://www.android.com/intl/it_it/auto/`.

[8] Feng Guo and Danni Lu. How many crashes does cellphone use contribute to? population attributable risk of cellphone use while driving. *Journal of Safety Research*, 82:385–391, 2022.

[9] Inayat Khan, Sanam Shahla Rizvi, Shah Khusro, Shaukat Ali, and Tae-Sun Chung. Analyzing drivers' distractions due to smartphone usage: evidence from autolog dataset. *Mobile Information Systems*, 2021(1):5802658, 2021.

[10] Muhammad Z Khan, Muhammad UG Khan, Omer Irshad, and Razi Iqbal. Deep learning and blockchain fusion for detecting driver's behavior in smart vehicles. *Internet Technology Letters*, 3(6):e119, 2020.

[11] Laith T. Khrais. Comparison study of blockchain technology and iota technology. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 42–47, 2020.

[12] Federico Montori, Marco Spallone, and Luca Bedogni. Texting and driving recognition leveraging the front camera of smartphones. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pages 1098–1103, 2023.

[13] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & information systems engineering*, 59:183–187, 2017.

[14] Eshed Ohn-Bar, Sujitha Martin, Ashish Tawari, and Mohan M. Trivedi. Head, eye, and hand patterns for driver activity recognition. In *2014 22nd International Conference on Pattern Recognition*, pages 660–665, 2014.

[15] Sajeesh Eringotkavil Raman and D. Venkatramaraju. Secured and transparent transactions using iota tangle distributed ledger technology in dairy supply chains. In *2024 International Conference on E-mobility, Power Control and Smart Systems (ICEMPS)*, pages 1–6, 2024.

[16] Wellington Fernandes Silvano and Roderval Marcelino. Iota tangle: A cryptocurrency to communicate internet-of-things data. *Future Generation Computer Systems*, 112:307–319, 2020.

[17] Tim Swanson. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. *Report, available online*, 28, 2015.

[18] Gheorghe-Daniel Voinea, Răzvan Gabriel Boboc, Ioana-Diana Buzdugan, Csaba Antonya, and George Yannis. Texting while driving: A literature review on driving simulator studies. *International Journal of Environmental Research and Public Health*, 20(5), 2023.

[19] Yang Xing, Chen Lv, Zhaozhong Zhang, Huaji Wang, Xiaoxiang Na, Dongpu Cao, Efstathios Velenis, and Fei-Yue Wang. Identification and analysis of driver postures for in-vehicle driving activities and secondary tasks recognition. *IEEE Transactions on Computational Social Systems*, 5(1):95–108, 2018.

[20] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. Movo: a dapp for dlt-based smart mobility. In *2021 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, 2021.

Appendici

Nell'appendice sono riportati alcuni frammenti di codice significativi che hanno avunto un ruole importante nell'implementazione del sistema descritto in questa tesi. Tra i codici presentati, troviamo l'implementazione della conversione da bit a trir, necessaria per supportare la rappresentazione ternaria dei dati richiesta dal protocollo IOTA. Inoltre è riportata la funzione di hash Curl-P-81 utilizzata nel processo di PoW. Infine l'implementazione della funzione `performParallelPow` che ha permesso di ridurre i tempi di ricerca del nonce in maniera significativa.

Listing 1: Codice per la conversione da bit a trit

```
// TryteValueToTritsLUT is a lookup table
// to convert tryte values into trits.
val TRYTE_VALUE_TO_TRITS_LUT = arrayOf(
    byteArrayOf(-1, -1, -1), byteArrayOf(0, -1, -1), byteArrayOf(1, -1, -1),
    byteArrayOf(-1, 0, -1), byteArrayOf(0, 0, -1), byteArrayOf(1, 0, -1),
    byteArrayOf(-1, 1, -1), byteArrayOf(0, 1, -1), byteArrayOf(1, 1, -1),
    byteArrayOf(-1, -1, 0), byteArrayOf(0, -1, 0), byteArrayOf(1, -1, 0),
    byteArrayOf(-1, 0, 0), byteArrayOf(0, 0, 0), byteArrayOf(1, 0, 0),
    byteArrayOf(-1, 1, 0), byteArrayOf(0, 1, 0), byteArrayOf(1, 1, 0),
    byteArrayOf(-1, -1, 1), byteArrayOf(0, -1, 1), byteArrayOf(1, -1, 1),
    byteArrayOf(-1, 0, 1), byteArrayOf(0, 0, 1), byteArrayOf(1, 0, 1),
    byteArrayOf(-1, 1, 1), byteArrayOf(0, 1, 1), byteArrayOf(1, 1, 1)
)
private fun mustPutTryteTrits(trits: Trits, offset: Int, value: Byte) {
    val idx = value - MIN_TRYTE_VALUE
    require(trits.size >= 3) {
        "The 'trits' array must have at least 3 elements."
    }
    trits[offset] = TRYTE_VALUE_TO_TRITS_LUT[idx][0]
    trits[offset + 1] = TRYTE_VALUE_TO_TRITS_LUT[idx][1]
    trits[offset + 2] = TRYTE_VALUE_TO_TRITS_LUT[idx][2]
}

fun encode(destination: Trits, source: ByteArray): Int {
    var numberOfTrits = 0
    for (i in source.indices) {
        val (t1, t2) = encodeGroup(source[i])
        mustPutTryteTrits(destination, numberOfTrits, t1)
        mustPutTryteTrits(destination, numberOfTrits +
            TRITS_PER_TRYTE, t2)
        numberOfTrits += 6
    }
    return  numberOfTrits
}

// encodeGroup converts a byte into two tryte values.
fun encodeGroup(byte: Byte): Pair<Byte, Byte> {
    val v = byte.toInt() +
        (TRYTE_RADIX / 2) * TRYTE_RADIX + (TRYTE_RADIX / 2)

    val quotient = v / TRYTE_RADIX
    val remainder = v % TRYTE_RADIX

    return Pair(
        (remainder + MIN_TRYTE_VALUE).toByte(),
        (quotient + MIN_TRYTE_VALUE).toByte()
    )
}
```

Listing 2: Codice per la funzione di hash Curl-P-81

```
const val HASH_LENGTH = 243
private const val STATE_LENGTH = 3 * HASH_LENGTH
const val NUMBER_OF_ROUNDS = 81
private val TRUTH_TABLE = byteArrayOf(1, 0, -1, 2, 1, -1, 0, 2, -1, 1, 0)

override fun absorb(
    trits: Trits,
    offset: Int,
    length: Int
): ICurl {
    var localOffset = offset
    var localLength = length
    do {
        System.arraycopy(trits, localOffset, state, 0,
            if (localLength < HASH_LENGTH) localLength else HASH_LENGTH)
        transform()
        localOffset = localOffset + HASH_LENGTH
        localLength = localLength - HASH_LENGTH
    } while (localLength > 0)

    return this
}

override fun squeeze(
    trits: Trits,
    offset: Int,
    length: Int
): Trits {
    var localOffset = offset
    var localLength = length

    do {
        System.arraycopy(state, 0, trits, localOffset,
            if (localLength < HASH_LENGTH) localLength else HASH_LENGTH)
        transform()
        localOffset = localOffset + HASH_LENGTH
        localLength = localLength - HASH_LENGTH
    } while (localLength > 0)

    return state
}

override fun transform(): ICurl {
    var scratchpadIndex = 0
    var prevScratchpadIndex = 0
    var truthTableIndex = 0
    repeat (NUMBER_OF_ROUNDS) {
        System.arraycopy(state, 0, scratchpad, 0, STATE_LENGTH)
        for (stateIndex in 0 until STATE_LENGTH) {
            prevScratchpadIndex = scratchpadIndex
            scratchpadIndex = if (scratchpadIndex < 365) scratchpadIndex +
                364 else scratchpadIndex - 365
            truthTableIndex = scratchpad[prevScratchpadIndex] +
                (scratchpad[scratchpadIndex].toInt() shl(2)) + 5
            state[stateIndex] = TRUTH_TABLE[truthTableIndex]
        }
    }
    return this
}
```

Listing 3: Codice per la funzione `performParallelPow`

```
@Volatile
private var nonceCounter = 0uL
 suspend fun performParallelPow(block: FullBlockWithTaggedDataPayload)
    : ULong = coroutineScope {
    val serializedMessage = serializeMessage(block)
    val powDigest = blake2b(serializedMessage)
    val powDigestTrits = b1t6Encode(powDigest)
    val curlFactory = { JCurl() }
    val maxCoroutines = 6
    val semaphore = Semaphore(maxCoroutines)
    val resultChannel = CompletableDeferred<ULong>()
    repeat(maxCoroutines) {
        launch {
            semaphore.withPermit {
                val curl = curlFactory()
                while (isActive) {
                    val nonce = getNextNonce()
                    val nonceBites = nonce.toLittleEndianByteArray(8)
                    val nonceTrits = b1t6Encode(nonceBites)
                    val inputTrits = powDigestTrits +
                        nonceTrits +
                        byteArrayOf(0, 0, 0)

                    curl.absorb(inputTrits)
                    val powHash = ByteArray(HASH_LENGTH)
                    curl.squeeze(powHash)
                    val trailingZeros = countTrailingZeros(powHash)
                    val message = serializedMessage + nonceBites
                    val score =
                        Math.pow(3.0, trailingZeros.toDouble()) /
                        message.size
                    if (score >= MIN_POW_SCORE) {
                        resultChannel.complete(nonce)
                        break
                    }
                    curl.reset()
                }
            }
        }
    }
    try {
        resultChannel.await()
    } finally {
        coroutineContext.cancelChildren()
    }
}
@Synchronized
private fun getNextNonce(): ULong {
    val current = nonceCounter
    nonceCounter++
    return current
}
```

Ringraziamenti

Vorrei ringraziare il Dott. Luca Sciullo e il Dott. Lorenzo Gigli, miei relatori, per la loro disponibilità e per avermi indirizzato durante lo sviluppo del progetto.

Un ringraziamento speciale va ai miei genitori, per il loro sostegno incondizionato e incoraggiamento durante la mia vita.

Un grande ringraziamento va a Fiorella che mi ha supportato e sopportato in tutti questi anni impegnativi, ci sei sempre stata nei momenti di bisogno dandomi immenso affetto, grazie iubi.

Ringrazio Juan per aver condiviso l'avventura del percorso scolastico in questi 16 anni, spronandomi a dare il massimo; è stato divertente.

Infine, un ringraziamento va ai miei amici, che con la loro presenza mi hanno sostenuto e celebrato insieme ogni traguardo.

Grazie a tutti.