UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Elettronica

Sviluppo e test di formati di pacchettizzazione per la trasmissione a bassa latenza in applicazioni di Earthquake Early Warning

Development and testing of packetization formats for low-latency transmission in Earthquake Early Warning applications

Candidato:
Tomas Rocchetti

Relatore:
Ing. Paola Pierleoni

Correlatore:
Dott.ssa Sara Raggiunto

Anno Accademico 2022-2023

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Elettronica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

In un contesto in cui la tempestiva identificazione e risposta a eventi sismici sono fondamentali, i sistemi di allerta precoce giocano un ruolo vitale nel garantire la sicurezza e la mitigazione dei danni a persone o cose. Nel contesto dell'Internet of Things (IoT), in cui sensori intelligenti sono ampiamente diffusi e interconnessi, l'utilizzo di formati di serializzazione e pacchettizzazione appropriati riveste un'importanza cruciale. Questa tesi si propone di esaminare attentamente i vari formati disponibili in letteratura, valutandone le prestazioni in termini di tempi di serializzazione e dimensioni dei pacchetti generati, al fine di proporre un nuovo formato ideale per l'utilizzo di sensori IoT all'interno dei sistemi di allerta precoce. L'obiettivo principale è quello di sviluppare un formato innovativo che ottimizzi l'efficienza della trasmissione dei dati, riducendo le dimensioni dei pacchetti e migliorando le prestazioni complessive del sistema.

Abstract

In a context where timely identification and response to seismic events are critical, early warning systems play a vital role in ensuring safety and mitigating damage to people or property. In the context of the Internet of Things (IoT), where smart sensors are widely deployed and interconnected, the use of appropriate serialization and packaging formats is of crucial importance. This thesis aims to examine in depth the various formats available in the literature, evaluating their performance in terms of serialization times and size of the generated packets, in order to propose a new format ideal for the use of IoT sensors within early warning. The main objective is to develop an innovative format that optimizes data transmission efficiency, reducing packet size and improving overall system performance.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi decenni si è assistito ad una notevole evoluzione delle reti di monitoraggio sismico, soprattutto nelle regioni ad alto rischio. Questi fenomeni naturali sono imprevedibili, e talvolta, portano a conseguenze catastrofiche.

Grazie ai progressi tecnologici, è possibile realizzare dispositivi sempre più precisi e sviluppare tecniche sempre più efficaci per il monitoraggio sismico in tempo reale. Ciò ha permesso alle reti sismiche di elaborare automaticamente i segnali registrati dai sensori e fornire stime rapide e affidabili dei parametri di un terremoto durante la sua fase di sviluppo. Questo processo è noto come "earthquake early warning" (EEW).
Il principio dell'EEW si basa sulla differenza di velocità di trasmissione che si ha tra le onde sismiche e la propagazione dell'informazione. La velocità di propagazione dell'informazione, infatti, può arrivare ad essere anche 70000 volte quella delle onde sismiche. Pertanto, se si è in grado di osservare il terremoto nelle immedita vicinanze del suo epicentro, si può pensare di mandare allerte a zone vicine prima che l'onda effettivamente arrivi.

Aumentare il numero di stazioni di monitoraggio nelle zona di origine dei terremoti può portare benefici sia in termini di precisione dei risultati, sia di tempi

di rilevamento. In passato, l'aumento della densità della rete comportava costi elevati per l'acquisto dei dispositivi e difficoltà nella gestione della comunicazione tra i vari nodi attraverso le reti di telecomunicazione tradizionali. Tuttavia, i recenti sviluppi nell'Internet of Things (IoT) hanno aperto nuove possibilità per l'applicazione dell'EEW. I dispositivi IoT possono essere integrati nelle reti sismiche esistenti, consentendo un monitoraggio più esteso del territorio in tempo reale e contribuendo a una rilevazione più tempestiva degli eventi sismici. Tuttavia, i dispositivi IoT presentano alcuni limiti legati alle risorse computazionali, di archiviazione e di energia disponibili. Per ridurre il carico di lavoro sui sistemi IoT, spesso si fa affidamento a piattaforme Cloud che offrono capacità di elaborazione e archiviazione virtualmente infinite per eseguire le operazioni computazionalmente più intensive. Le tecnologie basate sull'IoT si applicano in diversi contesti, il che implica l'utilizzo di protocolli di comunicazione in grado di adattarsi ad ogni esigenza. Uno dei protocolli ampiamente utilizzati nell'ambito dell'IoT, grazie alle sue caratteristiche, è MQTT (Message Queuing Telemetry Transport).

Nella prima parte di questo studio vengono quindi analizzati i principali formati che permettono di generare dati da inserire in un pacchetto MQTT. Tali formati, vengono inoltre messi a confronto con i formati più classici utilizzati nelle reti sismiche. Viene infine proposto una nuova soluzione, adattata al contesto dei dati sismici e ini grado di sfruttare al meglio le principali caratteristiche di MQTT.

Capitolo 2

Stato dell'Arte

2.1 Early Warning Systems

Nel corso del tempo, nella crosta terrestre, si accumulano tensioni. Queste tensioni vengono improvvisamente rilasciate generando un terremoto.

Tali tensioni si originano a causa del costante movimento delle placche tettoniche, esse, infatti, sono in costante moto a causa delle correnti convettive del mantello terrestre.

Quando le tensioni diventano troppo intense e superano la resistenza delle rocce lungo una faglia, tali rocce si rompono improvvisamente, liberando un'enorme quantità di energia sotto forma di onde sismiche.

Esistono due tipi di onde sismiche:

- **Onde primarie (P):** sono le prime ad arrivare, si muovono longitudinalmente comprimendo e dilatando le rocce nella direzione di propagazione. La velocità di propagazione si aggira tra i 4 e gli 8 $\frac{km}{s}$ e l'energia che trasportano è relativamente bassa, quindi vengono considerate poco distruttive.

- **Onde secondarie (S):** oscillano perpendicolarmente alla direzione di propagazione e causano movimenti che potrebbero portare alla frattura delle

rocce nella crosta. Esse viaggiano circa il 60% più lentamente delle onde primarie.

Le onde sismiche possono anche venire differenziate in base a quanto sono profonde. Vengono cconsiderate maggiormente distruttive le onde sismiche superficiali, queste hanno origine nel momento in cui le onde sismiche raggiungono la superficie terrestre e iniziano a muoversi con essa. Sono loro le principali cause dei danni derivanti dai terremoti.

Si possono considerare principalmente due tipi di onde di superficie: le onde di **Rayleigh** e le onde di **Love**.

Le onde di **Rayleigh** prendono il nome dal fisico John William Strutt, Lord Rayleigh, è stato lui a descriverle per la prima volta. Queste onde sono le più comuni e le più lente tra le onde di superficie. Si muovono in un movimento rotatorio e retrogrado simile a quello di un'onda che si estende sulla superficie di uno stagno quando si fa cadere un sasso sulla superficie dell'acqua. Le onde di Rayleigh causano principalmente movimenti verticali ed orizzontali del terreno. Sono responsabili di gran parte dei danni causati dai terremoti.

Le onde di **Love** prendono il nome dal matematico Augustus Edward Hough Love. Queste onde sono più veloci delle onde di Rayleigh e causano principalmente movimenti di taglio orizzontali nel terreno. A differenza delle onde di Rayleigh, le onde di Love non causano significativi movimenti verticali. Sono meno distruttive delle onde di Rayleigh, ma possono comunque causare danni significativi agli edifici.

In Figura 2.1 viene mostrata la forma d'onda del terremoto di magnitudo 5.5 registrato a Senigallia in data 9 Novembre 2023. La forma d'onda è stata ottenuta da una stazione dell'istituto nazionale di Geofisica e Vulcanologia (INGV). Sono facilmente distinguibili onda P e onda S.

Conoscendo il ritardo di arrivo dell'onda S dall'onda P è possibile determinare la distanza dall'epicentro del terremoto. Distribuendo numerose stazioni sismiche sul territorio è infine possibile sfruttare le misure ottenute da ciascu-

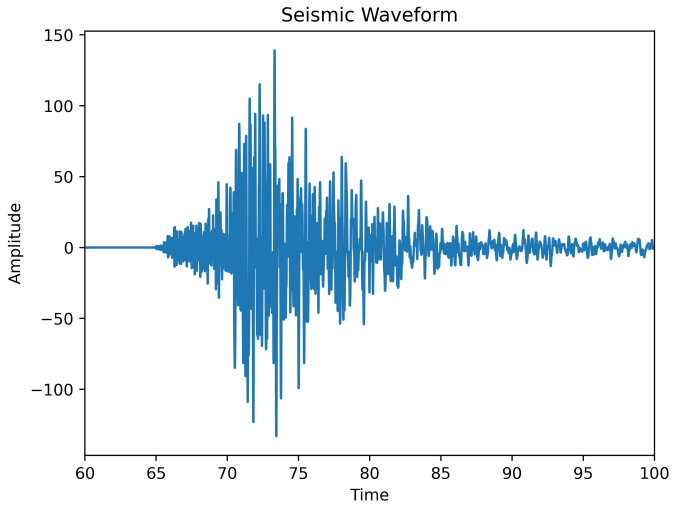

Figura 2.1: Forma d'onda dell'evento sismico registrato in data 09 Novembre 2022 a Senigallia

na di esse al fine di determinare la precisa locazione dell'epicentro mediante algoritmi di trilaterazione.

Oltre alla determinazione della posizione dell'epicentro, è possibile sfruttare il ritardo di arrivo tra due onde anche per sviluppare sistemi di allerta rapida. Gli Eartqake Early Warning Systems.

Si può cosí anticipare di numerosi secondi l'arrivo dell'onda distruttiva. Inoltre, ammesso di essere in gradi di trasmettere informazioni tramite la rete internet in maniera sufficientemente veloce, è anche possibile avvertire siti distanti dall'epicentro prima che l'onda effettivamente arrivi.

L'obiettivo principale di un sistema di early warning è quindi quello di fornire un avviso tempestivo alle persone, consentendo loro di prendere misure di sicurezza e ridurre al minimo i rischi. Un EEWS risulta molto utile anche nel caso in cui si vogliano bloccare in maniera autonoma sistemi ad alto rischio, come le forniture di gas.

Viene ora descritto in linea di massima il principio di funzionamento di un EEWS:

- **Rilevazione**: Le reti di sensori sismici rilevano l'onda sismica iniziale che si propaga attraverso la crosta terrestre. Tali sensori possono essere posizionati in varie località.

- **Analisi dei dati**: I dati raccolti dai sensori vengono inviati a un centro di elaborazione, dove vengono analizzati per determinare la magnitudo e l'epicentro del terremoto in corso. In base a questi dati, viene stimato il tempo di arrivo delle onde sismiche alla posizione di interesse.

- **Generazione dell'allerta**: Una volta che l'analisi dei dati conferma la presenza di un terremoto e ne viene stimato il tempo di arrivo, viene generato un avviso o un allarme che viene trasmesso agli utenti del sistema di early warning. Questi avvisi possono essere inviati attraverso numerosi mezzi.

- **Ricezione dell'allerta**: Le persone o gli enti che ricevono l'allerta precoce devono essere in grado di gestirla e di agire in maniera piuttosto tempestiva. Si possono quindi adottare misure di sicurezza come evacuare gli edifici.

In conclusione l'Earthquake Early Warning System (EEWS) non è quindi una previsione dell'accadimento di un terremoto, dato che l'evento è già avvenuto, ma dell'intensità con la quale l'evento giungerà ad un determinato sito [5].

2.1.1 Tipi di EEWS

Esistono principalmente due approcci per sviluppare un sistema di early warning: Regionale e on-site.

- L'EEWS **Regionale** moniotra l'attività sismica in una vasta area geografica, utilizzando numerosi sensori sismici distribuiti omogeneamente sul territorio. Un EEWS di questo tipo può fornire avvisi anticipati di

numerosi secondi pirima dell'arrivo del sisma. Può essere utile per mettere in sicurezza automaticamente alcuni sistemi critici, come quello della distribuzione del gas.

- L'EEWS **on-site** misura le onde sismiche in una specifica locazione, e si basa soltanto sulla differenza di arrivo tra onda P e onda S, è in grado di generare un allarme soltanto pochi istanti prima del sisma[4] [7].

2.1.2 EEWS Nel mondo

Gli EEWS (Early Earthquake Warning Systems) sono stati sperimentati in diverse parti del mondo, inclusa l'Italia. Paesi come il Giappone e gli Stati Uniti hanno avviato programmi di diffusione capillare delle allerte sismiche attraverso i media e hanno promosso campagne di educazione e informazione alla popolazione.

Il Giappone, ad esempio, ha implementato l'Early Warning Broadcast nel 2008, consentendo la diffusione immediata dei messaggi di allerta tramite i mass media. Negli Stati Uniti, il progetto "Shake Alert" dell'USGS (United States Geological Survey) sta sperimentando l'EEW in California da diversi anni, utilizzando la rete sismica integrata CISN. Questo progetto mira a verificare e testare l'allerta anticipata lungo la costa occidentale, che è minacciata dai terremoti lungo la famigerata faglia di San Andreas, nonché nella zona di subduzione della Cascadia.

Anche l'Unione Europea ha finanziato progetti come SAFER e REAKT negli ultimi dieci anni, con l'obiettivo di sviluppare e sperimentare gli EEWS nelle principali zone sismiche dell'area Euro-Mediterranea. In Italia, è stato implementato un prototipo di sistema di Early Warning Regionale chiamato PRESTo, sviluppato presso il Dipartimento di Fisica dell'Università di Napoli Federico II, che attualmente funziona a scopo di ricerca nel sud Italia.

Queste iniziative hanno l'obiettivo comune di fornire avvisi tempestivi e diffondere informazioni di allerta per ridurre i danni e garantire la sicurezza delle persone durante i terremoti.

2.1.3 PRESTo

Il sistema di allerta PRESTo (PRobabilistic and Evolutionary eaRthquake Early Warning SysTem) è un prototipo di sistema di allerta sismica in Italia, sviluppato presso il Dipartimento di Fisica dell'Università di Napoli Federico II. PRESTo è stato progettato per fornire un avviso anticipato di terremoti imminenti nella regione del Meridione. [9]

Il sistema PRESTo si basa su un'approccio probabilistico ed evolutivo per generare gli avvisi sismici. Utilizza una rete di sensori distribuiti nella regione monitorata per rilevare i segnali sismici. I dati sismici provenienti dai sensori vengono elaborati in tempo reale per valutare e stimare i parametri chiave dell'evento sismico in corso.

L'elaborazione dei dati sismici da parte di PRESTo consente di stimare la magnitudo del terremoto, la sua posizione e la sua evoluzione nel tempo. Sulla base di queste informazioni, il sistema genera un avviso anticipato che può essere trasmesso ad alcuni utenti, come operatori di protezione civile, enti pubblici, aziende e istituzioni, al fine di prendere provvedimenti adeguati e attivare misure di protezione e sicurezza.

PRESTo utilizza un approccio evolutivo per migliorare continuamente le sue capacità di previsione. Il sistema viene costantemente addestrato e calibrato utilizzando dati storici e informazioni sismiche in tempo reale. Ciò consente al sistema di adattarsi e migliorare nel tempo, rendendolo più affidabile ed efficiente nella generazione di avvisi sismici.

È importante sottolineare che PRESTo è ancora in fase di sviluppo e sperimentazione come prototipo. Attualmente, viene utilizzato a scopo di ricerca per valutare la sua efficacia e affinare le sue capacità di allerta sismica. Tuttavia, il sistema PRESTo rappresenta un passo significativo nell'implementazione di un

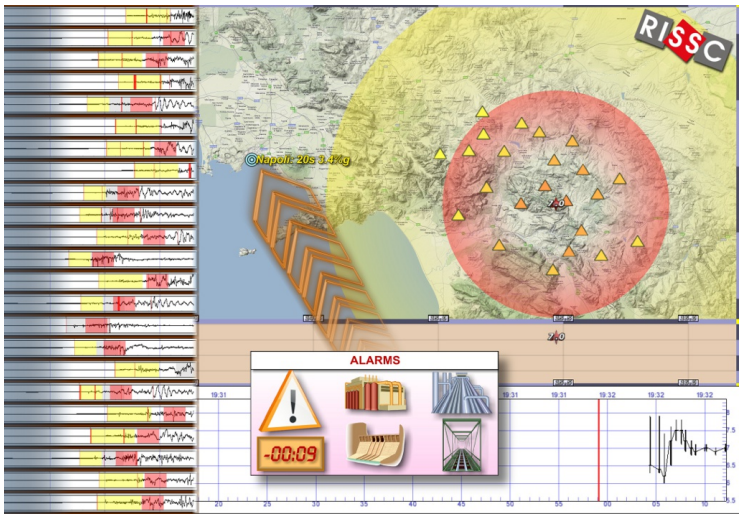

Figura 2.2: Interfaccia del sistema PRESTo

sistema di allerta sismica in Italia e potrebbe fornire un contributo importante alla gestione dei terremoti e alla protezione delle persone e delle infrastrutture nelle zone ad alto riischio sismico [9].

Il sistema sta funzionando sulla ISNet, una rete composta da 30 stazioni che, in tempo reale, trasmettono i segnali registrati a dei sistemi di controllo dedicati. Queste stazioni sono organizzate in sub-nets. Ogni sub-net è composta da un massimo di 6 o 7 stazioni connesse a dei Local Control Centers (LCC), connessi a loro volta a un Network Control Center (NCC). Il sistema a livello architetturale è basato su 5 moduli.

- **Acquisizione dei dati**: Il sistema PRESTo si collega ai server presenti in ogni Local Control Center (LCC) utilizzando il protocollo SEEDLink. I dati sismici vengono acquisiti dalle stazioni e viene assegnato un fattore di qualità a ciascuna stazione.

- **Picking della fase P ed associazione**: Utilizzando un algoritmo Filter-Picker, viene effettuato il picking della fase P sulla componente verticale dell'accelerazione. I tempi di arrivo della fase P vengono memorizzati e confrontati tra loro per determinare la coerenza temporale.

- **Localizzazione**: Utilizzando l'algoritmo RTLoc, si stima la posizione dell'ipocentro dell'evento sismico basandosi sulle stazioni che hanno rilevato la fase P. Vengono utilizzate griglie precalcolate contenenti modelli di velocità e tempi di propagazione.

- **Stima della magnitudo**: Le componenti dell'accelerazione vengono filtrate e viene calcolato lo spostamento del suolo. Utilizzando l'algoritmo RTMag, si stima la densità di probabilità del valore di magnitudo dell'evento sismico.

- **Stima dell'accelerazione di picco verso punti target**: Utilizzando leggi di attenuazione, si stimano le accelerazioni di picco più probabili ad una certa distanza dall'epicentro dell'evento sismico.

2.2 SEED

SEED è un formato standard internazionale utilizzato per scambiare dati sismologici digitali. È stato creato appositamente per la comunità di ricerca sismica, al fine di facilitare lo scambio di dati grezzi delle scosse tra le diverse istituzioni. Questo formato è progettato per rappresentare dati digitali raccolti in un punto nello spazio e a intervalli di tempo regolari.

L'utilizzo di SEED è di grande aiuto per i sismologi che registrano, condividono e utilizzano dati sismologici. Grazie alla sua standardizzazione, SEED semplifica e rende più precisi i processi di trasmissione, ricezione ed elaborazione dei dati sismici [2].

Il formato SEED è frutto della collaborazione di numerosi sismologi e informatici. Questo grande team ha dedicato la propria esperienza alla realizzazione di formati speciali per la distribuzione di dati sismici. Hanno operato in modo da garantire la massima compatibilità fra i diversi sistemi. Inoltre, hanno collaborato con organizzazioni standard come l'Organizzazione internazionale per le norme (ISO), l'Istituto nazionale di standard americano (ANSI) e altri enti del settore.

Il risultato di questo lavoro è stato il formato SEED, che è in grado di soddisfare le esigenze di tutte quelle persone e istituzioni coinvolte nella raccolta, registrazione, trasmissione e lettura di dati sismici. Il formato SEED si basa su alcune assunzioni comuni a tutti i formati di scambio di dati sismici digitali attualmente in uso nelle reti. Ad esempio, SEED si basa sull'utilizzo del Byte a 8 bit, comune alla maggior parte delle architetture informatiche sin dalla loro origine.

Il formato SEED è stato progettato per essere general purpose, quindi in

grado di gestire numerosi tipi di dati. Tuttavia, questa generalità comporta una certa complessità del formato. Il formato SEED è auto-definitorio, il che significa che ogni elemento o dato include tutte le informazioni necessarie per la sua interpretazione. Ciò semplifica l'elaborazione automatica dei dati. Il formato è anche robusto, in quanto include informazioni incorporate che consentono di recuperare i dati in caso di errori di registrazione o trasmissione.

SEED è molto efficiente in termini di spazio di archiviazione e tempi di elaborazione, riducendo al minimo lo spreco di risorse. Il formato è inoltre portatile, ovvero può essere letto facilmente da qualsiasi computer disponibile commercialmente che utilizzi standard comuni. Inoltre, il formato è leggibile da computer, il che significa che può essere facilmente elaborato e interpretato da software. È anche flessibile e in grado di supportare futuri utilizzi non ancora previsti, grazie all'inclusione di informazioni ausiliarie estendibili [2].

2.2.1 Organizzazione del SEED

Nonostante il formato SEED sia principalmente nato per lo scambio di dati sismici tra le istituzioni, esso viene anche impiegato per altri scopi. Infatti le stazioni sismiche che raccolgono ed inviano dati, spesso utilizzano il SEED come standard.

Il SEED trova applicazione anche in ambiti relativamente esterni dalla cerchia forme di onde sismiche, infatti viene utilizzato anche per registrare i dati di movimenti della crosta terrestre analizzati dai GPS.

Possiamo suddividere il SEED in 2 parti:

- **miniSEED**, è la parte contenente il succo dell'informazione, ovvero i Bytes rappresentanti la forma d'onda.

- **DatalessSEED**, è il complementare del miniSEED e contiene le coordinate geografiche e le informazioni sugli strumenti spesso necessarie per elaborare i dati delle serie temporali. Un volume [1] dataless può contenere un'intera e completa cronologia dei metadati per una o più reti e stazioni. Un volume dataless non contiene valori di serie temporali.

2.3 miniSEED

Come anticipato, il miniSEED è componente fondamentale dello standard SEED che viene utilizzato per lo scambio e l'archiviazione dei dati delle serie temporali. Oltre all'identificazione delle serie temporali e ai semplici indicatori di stato, miniSEED include un numero molto limitato di metadati. In particolare, non vengono inclusi le coordinate geografiche, le informazioni sulla risposta e altre informazioni necessarie per interpretare i valori dei dati. Solo facendo questi numerosi tagli è possibile far scendere la dimensione dell'header del pacchetto ad appena **48Bytes**.

Le serie temporali vengono memorizzate come record generalmente indipendenti e di lunghezza fissa, ognuno dei quali contiene un piccolo segmento di valori di serie contigui. Per ricostruire serie temporali più lunghe e contigue dai segmenti dei record di dati, è necessario utilizzare un software di lettura miniSEED. Le lunghezze comuni dei record sono di 512 byte se si lavora con trasmissione di flussi in tempo reale e 4096 byte se si vuole archiviare l'informazione, mentre altre lunghezze dei record vengono utilizzate per applicazioni piuttosto specifiche.

Esistono diverse librerie di programmazione che supportano la lettura e la scrittura semplici dei dati miniSEED senza conoscere i dettagli del formato. La libreria `libmseed` è ampiamente utilizzata e supportata dall'IRIS DMC [2].

Un pacchetto miniSEED è generalmente suddivisibile in 3 sezioni:

[1] Un volume SEED può essere un file o un singolo file `*.seed` o un gruppo di files `*.seed`

- **Header**, è grande `48 Bytes` e contiene tutte le informazioni caratteristiche della serie temporale, come frequenza di campionamento, data e ora di inizio, numero di campioni, ecc.

- **Blockette**, sono 2 blocchetti da `8 Bytes` ciascuno, quindi `16 Bytes` in totale. Contengono le informazioni riguardo la codifica delle serie temporali, l' eventuale algoritmo di compressione utilizzato e l'offset da cui iniziare a leggere.

- **Serie Temporale**, contiene la reale informazione riguardo la forma d'onda sismica. Queste informazioni posso essere codificate sia in numerosi formati classici come `INT8`, `INT16`, `INT32`, `FLOAT`, ma possono anche essere codificate sfruttando degli algoritmi di compressione differenziali, come `STEIM1` o `STEIM2`. Questi ultimi verranno successivamente analizzati.

Nelle figure 2.4, 2.3 qui di seguito vengono raffigurate le strutture di pacchetti miniSEED e DatalessSEED in maniera piuttosto dettagliata.

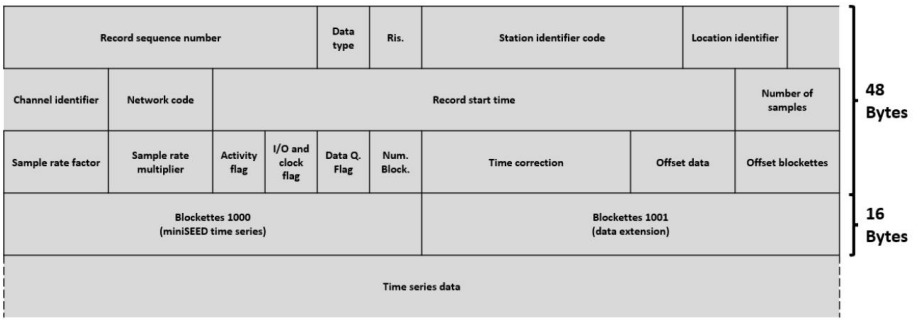

Figura 2.3: Forma di un pacchetto miniSEED visto nel dettaglio

Figura 2.4: Differenziazione di pacchetti MiniSEED e DatalessSEED

2.4 SEEDLink

2.4.1 Struttura di una rete per la trasmissione di dati sismici

Nel mondo, la rete adibita alla trasmissione e distribuzione dei dati sismici è molto diffusa, essa permette di fornire dati sismici a chiunque li richieda [Figura 2.5] [10].

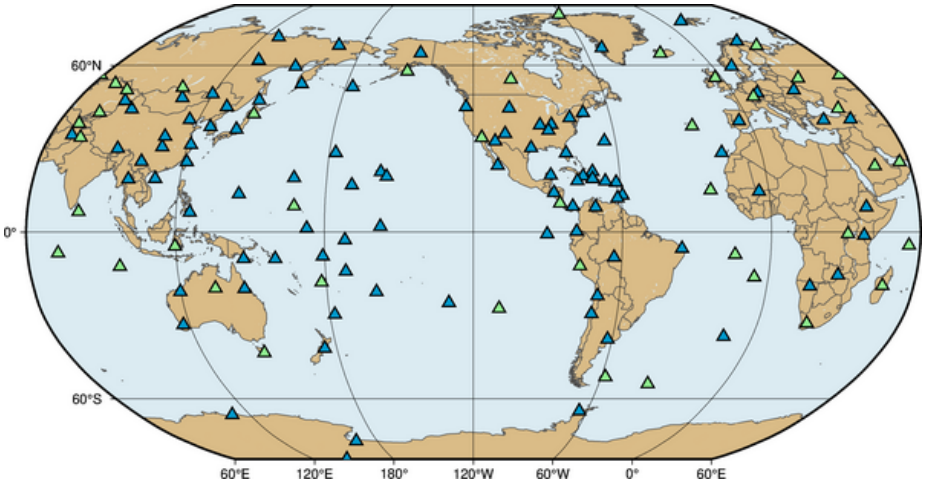

Figura 2.5: Distribuzione della rete di monitoraggio sismico nel mondo

La rete sismica in: Station, Network, Channel e Location. I nomi di questi eleementi sono utili a produrre i metadati da inserire nel pacchetto SEED.

- **Station**, si riferisce a una posizione fisica in cui sono presenti vari dispositivi di rilevazione. Questi nomi vengono generalmente abbreviati con cinque caratteri o meno. Storicamente, le stazioni che terminavano con la lettera O nella Rete Globale di Sismografi (GSN) erano stazioni adottate dall'Osservatorio di Ricerca Sismica e rappresentavano sensori installati ad elevata profondità. Questa convenzione non viene più mantenuta. Si noti anche che la GSN non è una rete secondo la nomenclatura SEED (viola la regola delle due o meno lettere) ed è composta da diverse altre

reti, come le reti II e IU. Ogni volta che una stazione viene spostata di più di 1 km dalla sua posizione iniziale, riceverà un nuovo nome, il quale viene registrato presso il Centro Sismologico Internazionale. Ad esempio, la stazione South Pole, in Antartide (SPA), è stata spostata in un sito vicino più silenzioso e ribattezzata Quiet South Pole Antartide (QSPA).

- **Network**, sono un agglomerato di *stations*. Le reti sono abbreviate con un codice di rete composto da una o due lettere assegnato dalla Federazione Internazionale delle Reti Sismiche Digitali (FDSN). Ad esempio, la porzione IRIS/USGS della GSN è **IU** (gestita dal Laboratorio Sismologico di Albuquerque), mentre la rete **II** è gestita da IRIS e dall'International Deployment of Accelerometers presso l'Università della California a San Diego. L'elenco delle reeti è consultabile da chiunque accedendo alla banca dati della FSDN.

- **Channel**, per una determinata serie temporale è un singolo flusso di dati. Le tre lettere del nome del canale includono la banda di frequenza, il tasso di campionamento, il codice dello strumento e il codice di orientamento. Un esempio potrebbe essere BNZ, per un accelerometro a larga banda (codice di banda B; 10-80 campioni/s) (codice dello strumento N) con asse verticale (codice di orientamento Z). È da notare notare che i codici di orientamento spesso non sono precisi. Nella maggior parte dei casi, ci saranno lievi errori di orientamento che sono descritti nei metadati della stazione. Pertanto, è sempre importante ruotare i dati, specialmente gli assi orizzontali, in base ai metadati e non fare affidamento esclusivamente sul codice di orientamento.

- **Location**, è composto da due lettere o numeri che aiutano a distinguere strumenti simili o canali con nomi simili nella stessa stazione. Questo codice diventa importante per stazioni come QSPA, dove ci sono più di cinque sismometri a banda larga vicini tra loro. Quindi, se si desiderano dati verticali a lungo periodo dal sensore principale nel pozzo (codice di posizione

00), il nome del dato a cui fare riferimento sarebbe QSPA.IU.LHZ.00. Il campo **Location** può essere importante anche quando ci sono flussi di dati simili con numerose frequenze di campionamento provenienti dallo stesso strumento. Ad esempio, i dati verticali a basso guadagno attivati (HNZ) così come i dati verticali continui a basso guadagno dallo stesso sensore possono essere distinti con diversi codici di posizione. In alcune stazioni, ci sarà un solo strumento e in queste situazioni talvolta gli operatori di rete utilizzeranno un codice di posizione vuoto o potrebbero utilizzare il codice di posizione vuoto per l'strumento principale. In questi casi, alcuni meccanismi di richiesta dati accetteranno "-" come codice di posizione vuoto.

In Figura 2.6 viene mostrata una chiara rappresentazione dell'organizzazione di una rete per lo scambio di dati sismici.

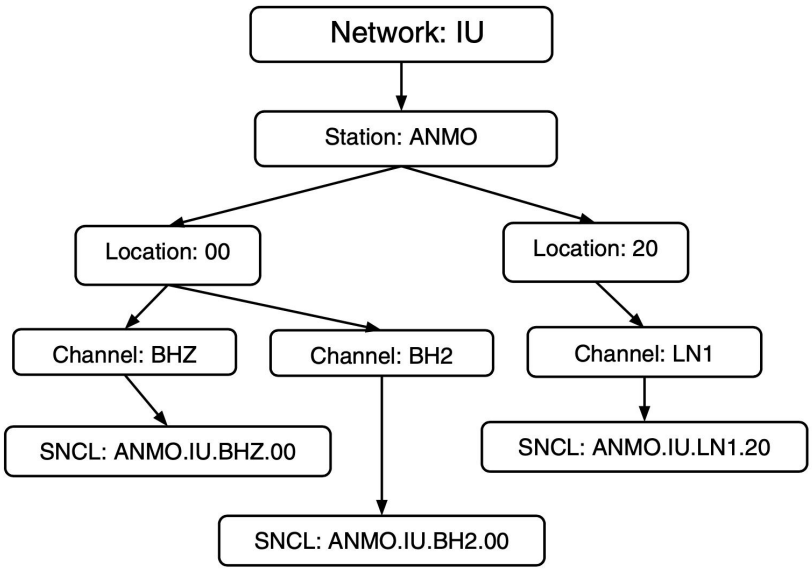

Figura 2.6: Diagramma di flusso per la struttura di una rete sismica con network, station, location e channel.

2.4.2 Descrizione del protocollo SEEDLink

Il protocollo SeedLink è un protocollo sviluppato sul livello applicativo della pila ISO/OSI, adatto per il trasferimento di serie temporali per applicazioni sismologiche e ambientali, con una latenza quasi real time. Una serie temporale è rappresentata da una sequenza di pacchetti di lunghezza variabile ed è identificata da una stazione e un ID di flusso.

La comunicazione SeedLink avviene tramite connessioni TCP/IP. La porta predefinita è la 18000 nel caso di connessione non crittografata e 18500 quando viene utilizzato TLS. Diverse serie temporali possono essere multiplexate in una singola connessione.

Una sessione SeedLink inizia con l'apertura di una connessione TCP/IP e termina con la sua chiusura. Lo stato di ogni sessione è indipendente dalle altre sessioni.

La sessione si sviluppa in due fasi principali:

- **Fase di negoziazione** (handshaking): Durante questa fase, il client invia comandi al server e riceve risposte a tali comandi. Il client può sfruttare questi comandi per settare delle configurazioni nel server. A seguito di ciò, la fase di trasferimento dati può iniziare.
- **Fase di trasferimento dati**: Durante questa fase, il client riceve una sequenza di pacchetti SEED contenenti dati. In questa fase, il server accetta solo i comandi `INFO` e `BYE` dal client e non deve accettare altri comandi.

Dopo aver stabilito la connessione, il server attende che il client avvii la fase di negoziazione inviando i comandi appropriati. Durante la fase di negoziazione, il client può inviare comandi di configurazione al server per adattarlo alle proprie esigenze. Successivamente, si può passare alla seconda fase.

I pacchetti SeedLink vengono inviati dal server al client durante la fase di trasferimento dati. Ogni pacchetto contiene un header di 8 byte costituito da

una stringa ASCII che inizia con le lettere `SL` seguite da una numerazione esadecimale specifica del server. È importante notare che più stazioni possono comunicare attraverso lo stesso canale TCP. Pertanto, il client deve verificare l'header miniSEED per identificare il mittente del pacchetto.

I dati vengono trasmessi come flusso continuo senza rilevamento degli errori o controllo del flusso, poiché queste funzionalità sono gestite dal protocollo TCP. Ciò assicura un trasferimento dati efficiente utilizzando hardware specifico.

Quando il server invia l'ultimo pacchetto dati, aggiunge la stringa `END` alla fine del pacchetto e attende che il client la riceva. In seguito, il client chiude la connessione.

La caratteristica chiave del protocollo SeedLink è la sua robustezza, in quanto è progettato per tollerare disconnessioni e riconnessioni casuali dei client senza perdita di dati. È eventualmente possibile recuperare dati persi fintanto che sono ancora presenti nei buffer del server. Per implementare un server SeedLink, è possibile utilizzare un'applicazione come Ringserver.

2.4.3 Ringserver

Ringserver è un buffer circolare basato su TCP che è stato progettato per lo streaming di dati in pacchetti. Le caratteristiche principali sono la scalabilità e la stabilità. Questa implementazione è ampiamente utilizzata per il trasporto di pacchetti contenenti serie di dati temporali in quasi-realtime. Il buffer circolare è implementato con un'architettura FIFO (First In First Out), il che significa che i pacchetti vengono elaborati nell'ordine in cui sono stati ricevuti, garantendo l'integrità e l'ordine temporale dei dati. Non è richiesto un formato dati specifico per il buffer circolare.

Ringserver supporta vari protocolli basati su TCP, tra cui DataLink, SeedLink e HTTP/WebSocket. Il server può essere configurato direttamente da riga di comando o tramite un file di configurazione specifico chiamato `ringserver`. In molti casi, entrambi i metodi vengono utilizzati per personalizzare il comportamento del server in base alle esigenze specifiche dell'applicazione.

Anche se l'architettura di base di Ringserver è generica, sono state apportate alcune aggiunte per supportare il protocollo SeedLink. In particolare, il server può essere configurato per servire i dati in pacchetti miniSEED da 512 byte attraverso SeedLink. Tuttavia, questa funzionalità deve essere abilitata, poiché l'ascolto delle connessioni SeedLink non è attivo di default. Ciò consente ai client di connettersi al server e sottoscriversi a flussi di dati specifici inviati da altri client, utilizzando l'identificativo univoco associato a ciascun pacchetto nel buffer circolare.

Un vantaggio importante di Ringserver è la possibilità, per i client, di selezionare il punto di partenza del buffer circolare in modo univoco. Questo è particolarmente utile nel caso di connessioni interrotte, poiché consente al client di riprendere lo streaming dei dati dal punto in cui si era interrotto, senza alcuna perdita di informazioni.

In conclusione, Ringserver fornisce una solida infrastruttura per il trasferimento e lo streaming di dati temporali, offrendo scalabilità, stabilità e la possibilità di gestire connessioni interrotte senza compromettere l'integrità dei dati.

2.5 STEIM Encoding

Gli algoritmi di compressione per i dati sismici **STEIM**, sono nati da un lavoro svolto ad Harvard nei primi anni '80 per costruire un sismografo digitale "a banda molto ampia" in grado di registrare la maggior parte dei segnali sismici in un singolo flusso di dati. Il primo sistema includeva un nuovo tipo di convertitore A/D, che mostrava una gamma dinamica di oltre 24 bit binari. Era necessario un nuovo formato di registrazione per rappresentare questo tipo di output senza perdere le informazioni presenti nel segnale a banda larga e senza supporti di archiviazione di elevate dimensioni. Senza compressione, i dati binari a 32 bit per una stazione sismica a 3 assi che campiona continuamente a 20Hz superano i 20 Mb giornalieri. Canali di dati aggiuntivi, come i dati rilevati con una frequenza di campionamento più elevata e i dati con una frequenza di campionamento inferiore continua, potrebbero spingere il volume a quasi 30 Mb/giorno. Una rete di 100 stazioni potrebbe quindi produrre 3Gb di dati al giorno. Questo rate, notevole anche per i giorni odierni, era assolutamente ingestibile nel 1984.
La tecnica di compressione, pertanto, doveva:

- rappresentare almeno un range dinamico di 28 bit.
- essere applicabile in generale e quindi senza perdita di informazioni.
- essere molto veloce, non richiedendo operazioni floating-point intensive, delle quali molti sistemi di registrazione, ancora oggi, non sono capaci.
- essere facile da comprendere e implementare.
- avere pochi stati e poter essere facilmente testata.
- ridurre in modo significativo il volume medio globale di dati sismici che transitano continuamente sulla rete.
- ridurre sufficientemente il volume dei dati per consentire una valutazione efficace dei dati compressi, almeno con un tasso superiore al "tempo reale" [6].

Livello 1

In una serie temporale di interi a 32 bit, ogni campione di dati composto da quattro byte. Una tale serie temporale che rappresenta i dati prodotti da un sismometro in condizioni normali, fuori dall'attività sismica, è di solito altamente correlata, ovvero ogni campione di dati è altamente prevedibile dati i pochi campioni precedenti. Inoltre, l'andamento della serie temporale tende ad avere una frequenza bassa rispetto al tasso di campionamento, quindi le differenze tra campioni consecutivi sono generalmente piuttosto piccole rispetto a un numero a 32 bit a scala completa. In effetti, tali differenze richiedono di rado più di quattro o cinque bit per essere rappresentate con la stessa precisione che si ha quando si usano i 32 bit. Queste differenze, quindi, possono essere facilmente rappresentate come quantità di 1 byte senza alcuna perdita di informazione. Se facciamo ciò, possiamo comprimere significativamente i dati e ridurre lo spazio necessario per memorizzarli.

Tuttavia, un'attività sismica significativa può generare differenze consecutive più grandi di una quantità a 8 bit. In tali rari casi (meno dell'1% del tempo), è possibile utilizzare una quantità a 2 byte o a 4 byte per rappresentare la differenza dei campioni.. Se un tale schema viene utilizzato per memorizzare i dati su un supporto di archiviazione, possiamo ottenere un rapporto di compressione superiore a 3,5 a 1, rispetto alla memorizzazione di tutti i campioni di dati originali nel formato a 32 bit. [3].

Livello 2

Nel 1990 il metodo è stato esteso con il **Livello 2**, che è molto simile al Livello 1, con l'aggiunta di molte altre possibili rappresentazioni di differenze binarie. Il livello 2 può rappresentare gruppi di differenze di 4,5,6,8,10,15 e 30 bit. L'ulteriore flessibilità nella rappresentazione migliora l'efficienza della compressione fino all'80% senza un notevole aumento del tempo di CPU. Molte stazioni sul campo sono aggiornate già dal 1991, ma poche hanno abilitato la funzione a

causa del ritardo nelle strutture di raccolta centrale per la gestione del formato [6].

Livello 3

Sempre nel 1990, venne proposto un terzo livello di compressione che essenzialmente espandeva le tecniche dei livelli 1 e 2 includendo:

- La sostituzione dei modelli comuni di compressione delle mappe di bit con codici di un solo byte, riducendo di circa il 5% il costo in termini di dimensioni. Questo viene chiamato *Flag Squeezing*.
- La suddivisione in due gruppi di 32 bit per consentire una rappresentazione di 5 differenze consecutive di 12 bit e 3 differenze di 20 bit

Il Livello 3 non venne implementato nelle stazioni della rete globale in quel periodo. Lo schema di codifica viene descritto di seguito, Figura 2.7 [6].

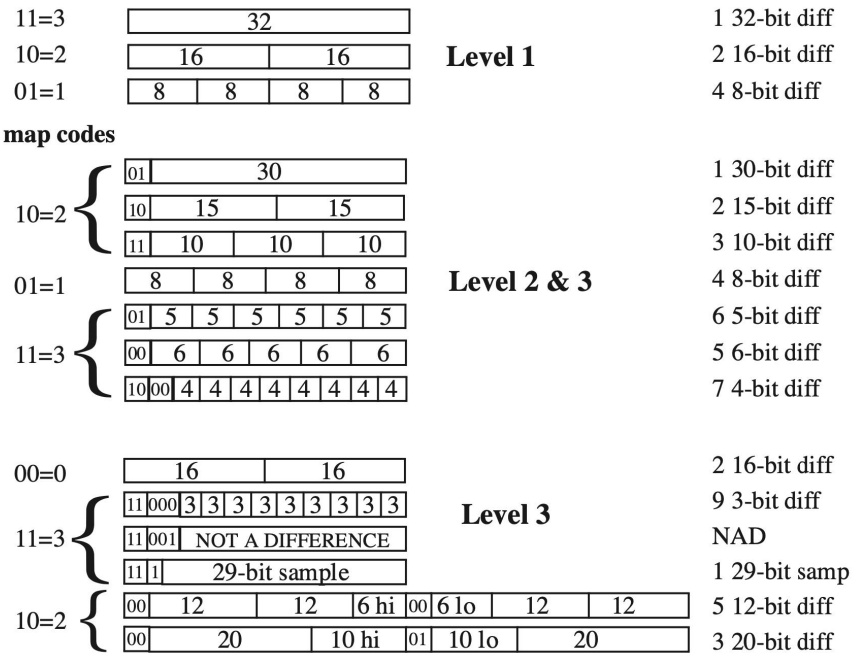

Figura 2.7: Rappresentazione dei pacchetti STEIM

2.6 Internet of Things

L'internet of things (IoT) è una rivoluzione tecnologica che rappresenta il futuro dell'informatica e della comunicazione. Il suo sviluppo dipende principalmente dalle numerose innovazioni tecnologiche fiorite nell'ultimo decennio in campi come la sensoristica o le nano tecnologie.

Non esiste una definizione univoca per l'Internet of Things che sia accettabile per l'intera comunità. In effetti, ci sono molti gruppi diversi tra cui accademici, ricercatori, professionisti, innovatori e sviluppatori che hanno definito a proprio modo il termine, sebbene il suo uso iniziale sia stato attribuito a Kevin Ashton, un esperto di innovazione digitale. Ciò che accomuna tutte le definizioni è l'idea che la prima versione di Internet riguardasse i dati creati dalle persone, mentre la versione successiva riguarda i dati creati dalle cose. La migliore definizione per l'Internet of Things sarebbe: "Una rete aperta e completa di oggetti intelligenti che hanno la capacità di auto-organizzarsi, condividere informazioni, dati e risorse, reagire e agire di fronte a situazioni e cambiamenti nell'ambiente" [12].

La prima applicazione di IoT venne fatta su di un distributore automatico di lattine nell'università di Carnegie Melon verso l'inizio degli anni '90. I programmatori lavoravano numerosi piani sopra al distributore, quindi svilupparono un sistema in grado di determinare se il distributore fosse pieno o vuoto. Furono così in grado di collegarsi al dispositivo da remoto e determinare se ci fossero delle bibite fresche. In questo modo potevano decidere se fare il viaggio per arrivare al distributore oppure no.

2.6.1 Requisiti per l'IoT

Per implementare con successo l'Internet of Things (IoT), i prerequisiti sono:

- Possibilità di espandere la rete in maniera arbitraria.
- Abilità nel funzionare in real time.

- Costo molto basso.
- Protezione per dati e privacy degli utenti.
- Efficienza energetica dei dispositivi.
- Accesso ad un sistema cloud aperto.

2.6.2 Architettura secondo ITU

Secondo le raccomandazioni dell'International Telecommunication Union (ITU), l'architettura delle reti dell'Internet of Things è composta dai seguenti livelli:

- Il livello di rilevamento/sensori (*Sensing Layer*)
- Il livello di accesso (*Access Layer*)
- Il livello di rete (*Network Layer*)
- Il livello di middleware (*Middleware Layer*)
- Il livello di applicazione (*Application Layer*)

Questi livelli sono simili al modello di riferimento Open Systems Interconnection (OSI) utilizzato nelle reti e nelle comunicazioni dati.[12]

2.6.3 MQTT

MQTT è un protocollo di messaggistica con standard OASIS per l'Internet of Things. È progettato secondo una logica publish/subscribe, è estremamente leggero, è ideale per connettere dispositivi remoti impiegando poco codice e una larghezza di banda di rete minima. MQTT viene utilizzato in un'ampia varietà di settori [8].

Sono numerosi i punti chiave che hanno permesso ad MQTT di ottenere un enorme successo:

- **Leggero ed efficiente**, Il codice necessario per sviluppare un client MQTT è minimo, e le risorse necessarie sono poche, questo permette di utilizzare

MQTT anche su microcontrollori. Inoltre gli headers dei messaggi sono brevi, così da ottimizzare la banda.

- **Affidabile**, in molte applicazioni IoT è indispensabile garantire il recapito dei messaggi. Per questo MQTT permette di scegliere tra 3 livelli di quality of service.
- **Bidirezionale**, MQTT permette sia la comunicazione da client a cloud che quella da cloud a client. Questo permette di diffondere in maniera rapida ed efficace messaggi broadcast a tutti i client connessi.
- **Supporto a reti inaffidabili**, MQTT può essere usato anche su reti poco affidabili. È nato per essere persistente e per riconnettersi nel minor tempo possibile.
- **Scalabile**, MQTT è scalabile e può essere utilizzato anche su milioni di dispositivi.
- **Sicuro**, Può cifrare i messaggi usando *TLS* e può autenticare i client utilizzando tecniche come *OAuth*.

2.6.4 IoT nell'earthquake early warning system

In sistemi di allerta rapida per terremoti il concetto chiave consiste nell'osservare le onde P di un terremoto in prossimità del suo epicentro, così avvisare tempestivamente le popolazioni vicine o avviare autonomamente sistemi di sicurezza.

Qualora si dovesse ricadere nel caso ideale di osservare il terremoto nelle immediate distanze dall'epicentro, Si potrebbero:

- Inviare avvisi anticipati di pochi istanti, facendo leva sul ritardo di arrivo tra le onde S e le onde P.

- Inviare segnali alle popolazioni o istituzioni più lontane. Dato che le onde sismiche viaggiano a velocità relativamente basse si potrebbe essere in grado di avvertire alcune zone con un anticipo di numerosi secondi.

Al fine di aumentare la possibilità di osservare un terremoto nelle immediate vicinanze dall'epicentro è intuibile pensare di diffondere sul territorio una elevata quantitatià di sensori.
Questa diffusione può essere resa possibile sfruttando dispositivi IoT. Essi, infatti, sono estremamente economici, efficienti e sono in grado di essere gestiti in grandi quantità. Inoltre, grazie a protocolli come MQTT è possibile far collaborare migliaia di nodi senza particolari sforzi.

2.7 Formati di serializzazione

La serializzazione è un processo che ci permette di convertire un oggetto in uno stream di bytes, fondamentale per permettere la memorizzazione di tale oggetto in un supporto di archiviazione o per la sua trasmissione in rete.
Lo scopo della serializzazione è quindi quello di far si che l'informazione contenuta nell'oggetto non venga alterata in fase di trasmissione o di archivio e che, successivamente, sia possibile deserializzare, e quindi ricostruire l'intero oggetto.

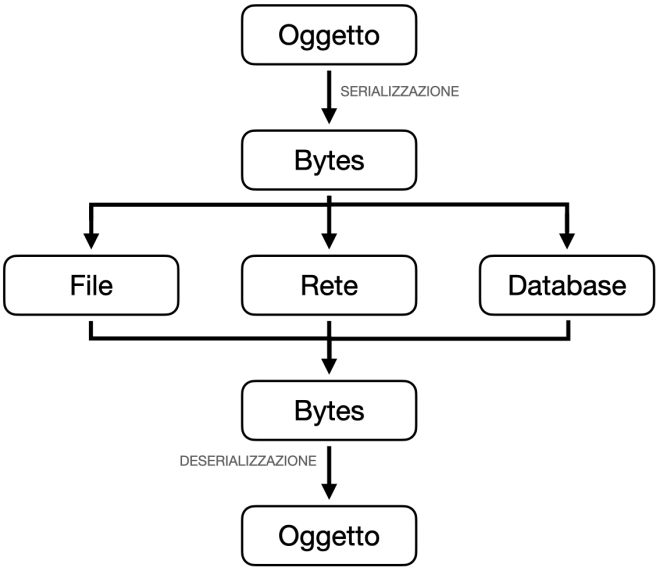

Figura 2.8: Flow chart che spiega in cosa consistono serializzazione e deserializzazione

Esistono numerosissimi formati di serializzazione, possiamo principalmente suddividerli in 2 categorie:

- **Leggibili dall'umano**, come XML, YAML, JSON.
- **Non immediatamente comprensibili dall'umano**, come MiniSEED, CBOR, Protobuf.

Nei seguenti sottocapitoli verranno analizzati uno ad uno i formati di serializzazione di interesse

2.7.1 JSON

JSON (JavaScript Object Notation) è un formato di serializzazione nato per lo scambio di dati. È decisamente facile da leggere e scrivere per gli esseri umani, mentre la sua sintassi è di facile analisi per le macchine. Si basa su un sottoinsieme del Linguaggio di Programmazione JavaScript, Standard ECMA-262 Terza Edizione - Dicembre 1999.
JSON è un formato completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C, come C, C++, Java, JavaScript, Perl, Python. Questa caratteristica fa di JSON un linguaggio ideale per lo scambio di dati.
JSON si basa su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, una struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste strutture dati sono universali, il che rende possibile l'implementazione di decoder ed encoder JSON in qualsiasi linguaggio di programmazione.
Le coppie nome/valore prendono il nome di oggetto, iniziano e finiscono con parentesi graffe e vengono realizzate come in Figura 2.9

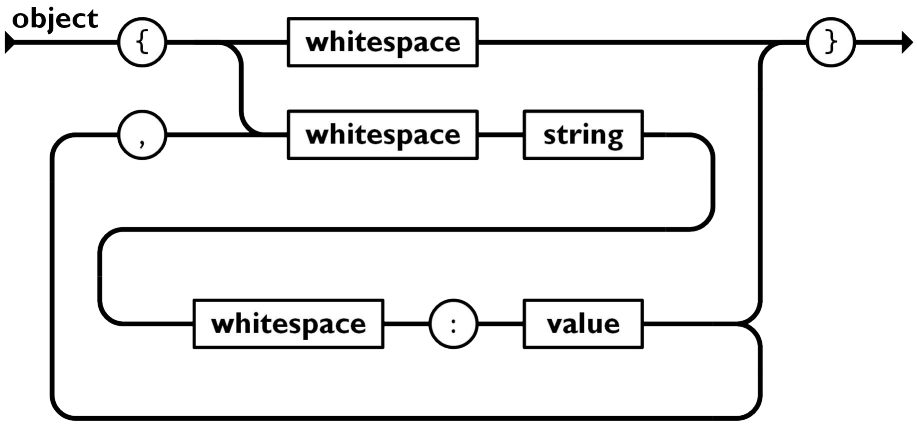

Figura 2.9: Flow chart che spiega la sintasi di un oggetto JSON

Un array JSON rappresenta una serie ordinata di valori, inizia e termina con delle parentesi quadre e viene generato come in Figura 2.10

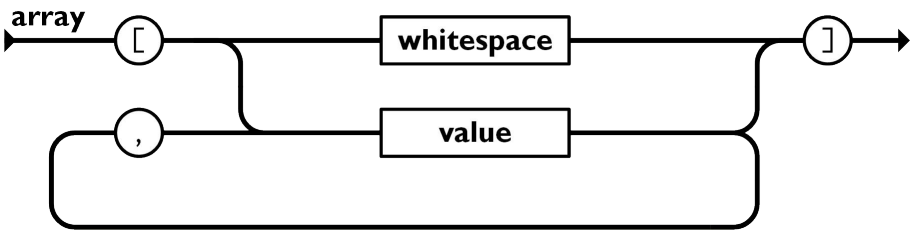

Figura 2.10: Flow chart che spiega la sintasi di un array JSON

Infine, un valore JSON, si può rappresentare come in Figura 2.11:

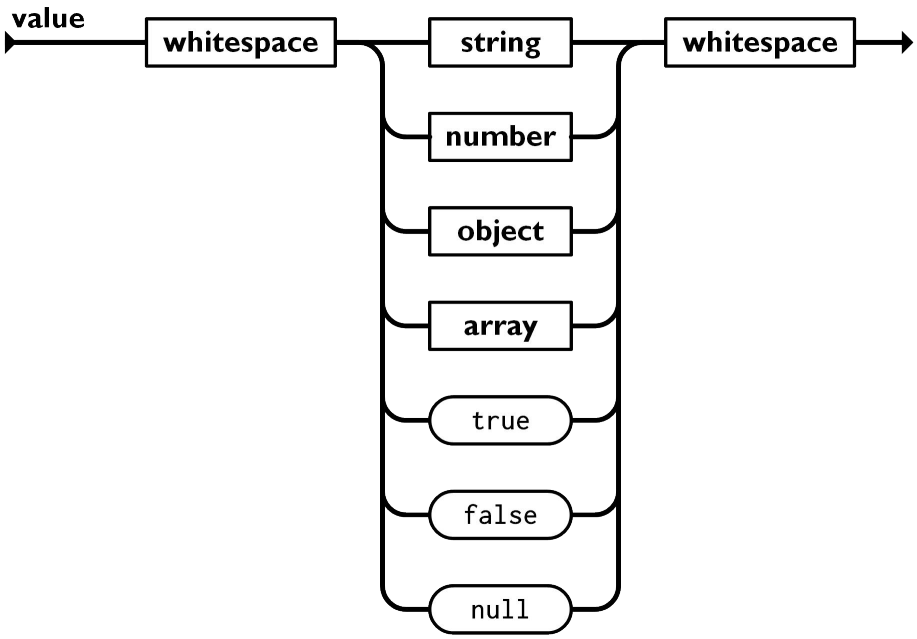

Figura 2.11: Flow chart che spiega la sintasi di un valore JSON

Viene mostrato ora l'esempio di un JSON:

```
{
    "nome": "Tomas",
    "cognome": "Rocchetti",
    "is_student": true,
    "numeri_telefonici": ["33333333333", "34444444444"]
}
```

2.7.2 CBOR

La Concise Binary Object Representation (CBOR) è un formato di dati che si pone l'obiettivo di avere di avere dimensioni di codice estremamente ridotte, dimensioni del messaggio minime ridotte ed espandibilità senza dover concordare una versione [1].

Definiamo ora i punti cardine del CBOR:

- La rappresentazione deve riuscire a codificare in modo assolutamente univoco i formati di dati più comuni utilizzati negli standard internet.
  - Deve rappresentare buona parte dei tipi e delle strutture di base utilizzando la codifica binaria. Le strutture supportate sono limitate a matrici e alberi. Loop e grafici non sono supportati.
- Il software di codifica e decodifica deve essere leggero e compatto, in modo da poter supportare anche sistemi con prestazioni, memoria e set di istruzioni del processore estremamente limitate.
- I dati devono poter essere decodificati senza la descrizione di uno schema, analogamente a quanto accade con il JSON, in cui i dati si auto descrivono.
- La serializzazione deve essere abbastanza compatta, nonostante ciò, la compattezza dei dati è secondaria alla compattezza del codice per codificare e decodificare.
- Il formato deve supportare tutti i tipi di dati supportati in JSON, per la conversione da e per JSON.
- Il formato deve essere estensibile, e l'estensione deve essere decodificabile dai decodificatori più vecchi.

Modelli di dati CBOR Come anticipato, CBOR descrive un modello esplicito di base per il formato dei dati. Il modello di base è comunque estensibile mediante l'inserimento dei cosí detti valori semplici e tag. Le applicazioni possono cosí costruire un sottoinsieme del modello generico per creare i loro modelli specifici.

Nel modello di dati di base, un elemento può essere uno dei seguenti:

- Un **intero** compreso tra -2^{64} e $2^{64} - 1$
- Un **valore semplice**, identificato da un numero compreso tra 0 e 255 ma distinto dal'intero con lo stesso valore.

- Un **valore a virgola mobile**.
- Una **sequenza di zero o più byte**, ovvero una stringa di codice.
- Una **sequenza di zero o più punti di codice Unicode**.
- Una **sequenza di zero o più elementi di dati**, ovvero un array.
- Una **mappatura**, ovvero una funzione matematica associa a zero o più elementi detti chiavi, ciascuno ad un elemento di dati, che potrebbe essere sia un valore che un array.
- Un **elemento di dati con tag**, che comprende il valore di un tag e il contenuto del tag, che potrebbe essere un elemento di dati.

In alcuni contesti potrebbe essere importante notare che i valori a virgola mobile e i valori interi sono distinti anche se contengono lo stesso valore.

Codifica CBOR Un elemento CBOR viene codificato (o decodificato) da (o in) una stringa di bytes. Un codificatore deve produrre soltanto modelli di dati codificati correttamente, mentre un decodificatore non deve restituire un dato quando incontra un oggetto non codificato correttamente. Per poter soddisfare questi requisiti il byte iniziale di ciascun elemento di dati codificato contiene sia informazioni sul tipo principale (i 3 bit di ordine superiore) sia informazioni aggiuntive (i 5 bit di ordine inferiore), come mostrato in Figura 2.12.

Figura 2.12: Raffigurazione della struttura del byte iniziale di un dato CBOR

Il valore delle informazioni aggiuntive descrive come caricare un "argomento":

- Se l'argomento è < 24 viene caricato il valore delle informazioni aggiuntive.
- Se l'argomento è 24,25,26 o 27 il valore del dato è contenuto negli 1,2,4 o 8 byte successivi.
- Se il tipo principale è 7 e il valore delle informazioni aggiuntive è 25 viene caricato un valore in virgola mobile.
- 28,29 e 30 sono riservati per future applicazioni.
- 31 indica un elemento non ben formato.

Si mostra infine un esempio di serializzazione CBOR.

```
a4                                      # map(4)
   64                                   #   text(4)
      6e6f6d65                          #     "nome"
   65                                   #   text(5)
      546f6d6173                        #     "Tomas"
   67                                   #   text(7)
      636f676e6f6d65                    #     "cognome"
   69                                   #   text(9)
      526f63636865747469                #     "Rocchetti"
   6a                                   #   text(10)
      69735f73747564656e74              #     "is_student"
   f5                                   #   true, simple(21)
   71                                   #   text(17)
      6e756d6572695f74656c65666f6e696369 #     "numeri_telefonici"
   82                                   #   array(2)
      6b                                #     text(11)
         33333333333333333333           #       "33333333333"
      6b                                #     text(11)
```

```
33343434343434343434          #       "34444444444"
```

2.7.3 ProtoBuf

Il Protocol Buffer (ProtoBuf), sviluppato da Google per la serializzazione dei dati strutturati, è un formato estensibile, indipendente dalla lingua e dalla piattaforma: come XML, ma più compatto, più veloce e più semplice. Si definiscono le strutture di dati una sola volta, a seguito di questo è possibile leggere e scrivere nelle strutture in qualsiasi linguaggio mediante codice opportunamente compilato.

Definizione di un messaggio Protobuf Per definire una struttura dati in Protobuf è sufficiente descrivere la struttura all' interno di un file `.proto` strutturato come nel seguente codice:

```
syntax = "proto3";


message Person {
    string nome = 1;
    string cognome = 2;
    bool is_student = 3
    repeated string numeri_telefonici = 4;
}
```

La prima riga definisce la versione di ProtoBuf utilizzata, in questo caso `proto3`. In seguito viene definita una struttura chiamata `Person` che ha 4 proprietà.
Per descrivere ogni proprietà si inizia definendo il tipo, che può essere:

- **double**.
- **float**.
- **int32**/**int64**, utilizzano la variable-lenght encoding, pertanto non sono efficienti nel momento in cui si voglia codificare interi negativi. In quel caso è consigliato utilizzare sint32 o sint64.

- **uint32/uint64**, utilizzano la variable-lenght encoding per ottimizzare le dimensioni del dato serializzato.
- **sint32/sint64**, utilizzano la variable-lenght encoding, come intuibile dal nome sono ottimizzati per codificare i signed int.
- **fixed32/fixed64**, interi di dimensione sempre pari a 4/8 Bytes, sono migliori degli uint32/uint64 se i calori da codificare sono superiori a $2^{28}/2^{56}$
- **bool**.
- **string**, deve sempre contenere un testo condificato UTF-8 o ASCII 7bit, non può essere più lungo di 2^{32}.
- **bytes**, può contenere una qualsiasi sequenza di bytes, a patto che non sia più lunga di 2^{32}.

Inserendo `repeated` davanti al tipo è possibile definire un array.
A seguito della definizione del tipo è possibile assegnare un nome di riferimento al dato.
Infine, al di la dell'= viene inserito un indice, univoco per ogni elemento della struttura. L'indice viene utilizzato come pratico metodo per individuare uno specifico elemento all'interno della struttura senza dover codificare l'intero nome sotto forma di stringa (come invece avviene in JSON).
L'indice può essere un intero compreso tra 1 e 536870911, è possibile scegliere qualsiasi valore all'interno di questo range a patto che ogni indice in una struttura sia univoco. ad eccezione di alcuni valori riservati, che vanno da 19000 a 19999. Tendenzialmente si preferisce scegliere valori bassi, questo perchè se si utilizza una codifica di tipo variable-lenght tali valori possono essere espressi da meno bit.
Ad esempio se scegliamo indici da 1 a 15 protobuf utilizzerà soltanto `4bit` per codificarne ciascuno.

Come studiato in [11], protobuf è in grado di generare messaggi di dimensioni drasticamente inferiori rispetto a formati di serializzazione più diffusi come

il JSON. Pertanto sembra essere un ottimo candidato per la trasmissione di informazioni attraverso i poco prestanti dispositivi per IoT.

Capitolo 3

Materiali, metodi e soluzione proposta

3.1 Il problema

Un terremoto inizia a manifestarsi a partire dalle onde P, poco intense e poco pericolose, per poi proseguire il suo sviluppo sotto forma di onde S.
Si vuole quindi ridurre al minimo L'Intervallo di tempo richiesto per la registrazione e l'invio delle informazioni contenenti le serie temporali da analizzare. Questo intervallo idealmente dovrebbe partire nel momento in cui si registrano onde P.
Tali tempistiche sono di vitale importanza nel momento in cui si vogliano avvertire le popolazioni ad alto rischio di danni prima dell'arrivo delle onde S.

3.2 Analisi del lavoro precedente

Il lavoro svolto è partito dall'analisi di un'altra ricerca. In quest'ultima, le tempistiche di serializzazione venivano misurate mediante degli script scritti in Python, Un linguaggio di programmazione molto potente ma poco efficiente e che non permette di avere una buona interazione con i livelli più bassi del sistema operativo.

Questo ha portato a produrre delle tempistiche di serializzazione con valori decisamente più alti rispetto a quelli attesi, pertanto è stato scelto di riscrivere l'intera suite[1] in C++.

Infine, è sorta una notevole problematica dovuta alle librerie miniSEED per il Python: per generare un pacchetto miniSEED era necessario salvarlo nella memoria di massa, incrementando in maniera non indifferente le tempistiche complessive [13].

[1] Una suite software è un insieme integrato di programmi informatici che vengono distribuiti e utilizzati insieme come un'unica soluzione. Questi programmi sono progettati per lavorare in sinergia e spesso condividono dati e funzionalità comuni.

3.3 Setup Dell'ambiente

L'intera suite di test è stata sviluppata in C++, l'IDE utilizzato è xCode. La scelta dell'IDE è stata dettata dalla praticità che ha lo sviluppatore con esso.

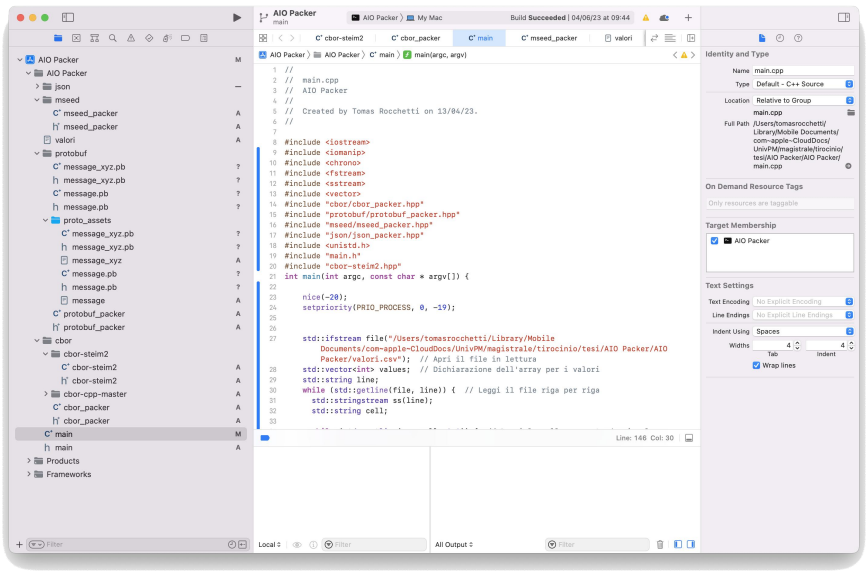

Figura 3.1: Interfaccia utente xCode

3.3.1 Programma principale

Veerrà, di seguito, illustrato il file del programma principale `main.c`, si noti che le istruzioni per il print[2] su terminale sono completamente esterne ai blocchi di codice di cui si misurano le tempistiche. La scelta di non includere tali log nelle tempistiche misurate è stata fatta in virtù del fatto che il print su seriale è un'operazione molto lenta, e avrebbe potuto invalidare i dati ottenuti.

All'inizio del programma, a seguito dell'inclusione delle varie librerie, vengono richiamate due istruzioni per dare massima priorità al processo:

```
1    nice(-20);
```

[2]visualizzazione di testo o risultati di un programma direttamente sulla console o sullo schermo del computer. Questa operazione consente di comunicare informazioni al programmatore o all'utente durante l'esecuzione di un programma.

```
    setpriority(PRIO_PROCESS, 0, -19);
```

Dopodiché viene eseguito il caricamento e fatto il parsing [3] del file `*.csv` contenente la forma d'onda di un terremoto.

```
    // Apri il file in lettura
    std::ifstream file("valori.csv");

    // Dichiarazione dell'array per i valori
    std::vector<int> values;
    std::string line;

    // Leggi il file riga per riga
    while (std::getline(file, line)) {
      std::stringstream ss(line);
      std::string cell;

      // Leggi le celle separate da virgola
      while (std::getline(ss, cell, ',')) {

        // Converte la stringa in un intero
        int value = std::stoi(cell);
        // Aggiungi l'intero all'array
        values.push_back(value);
      }
    }
```

Dopo aver caricato in RAM l'intero `*.csv`, il software entra in un loop lungo 2000 cicli.

Ad ogni ciclo `i` viene generato un pacchetto per ogni formato di serializzazione, e all'interno di esso vengono inseriti tutti i campioni della forma d'onda, da `0` fino ad `i`.

[3] processo di analisi e interpretazione di una sequenza di simboli conformi a una determinata grammatica o regola sintattica.

Vengono inoltre definite delle variabili comuni ad ogni formato, queste sono utili per definire i metadati [4] da inserire in ogni pacchetto:

```
int e = 42;
std::string t = "2012-01-01T00:00:00";
std::string i = "STATION_IV_PZ";
```

Ad ogni ciclo vengono monitorate le tempistiche di pacchettizzazione e le dimensioni del pacchetto generato, per poi effettuare un log di tutti i risultati. Utile per avere un file `*.csv` di riepilogo.

3.4 Svolgimento dei test

Nel capitolo seguente verranno proposti i test effettuati su numerosi formati di serializzazione comuni al fine di valutarne potenzialità e problemi. Verranno eseguiti 3 test:

- Test sulle tempistiche di serializzazione dei pacchetti, per valutare in che modo aumentano i tempi sulla base di quanti campioni vengono inseriti in ciascun pacchetto.
- Test sulle dimensioni di un pacchetto, per valutare in che modo variano le dimensioni complessive di un pacchetto sulla base dei campioni che vengono inseriti in ciascun pacchetto.
- Test sulla variazione nelle dimensioni di un pacchetto, per valutare in che modo varia la dimensione di un pacchetto a parità di numero di campioni, ma variando i valori di ogni campione.

Tra i formati analizzati sono presenti:

- CBOR.
- JSON.
- Protobuf.

[4] Dati utili all'interpretazione della serie temporale.

- MiniSEED INT32.
- MiniSEED STEIM2.

Viene inoltre proposto un test effettuato applicando la codifica STEIM2 al formato CBOR.

3.4.1 Adattamento della codifica STEIM al formato CBOR

Lo sviluppo del formato CBOR con codifica STEIM2 è stato fatto partendo dalla libreria MiniSEED ufficiale scritta in *C*, da questa sono state estratte in maniera completamente agnostica numerose funzioni necessarie per la generazione di dati STEIM2. In particolare:

```
1    int msr_encode_steim2 (int32_t *input, int samplecount, int32_t
      *output, int outputlength, int32_t diff0, uint16_t *
     byteswritten, int swapflag)
```

Questa funzione è il cuore della codifica STEIM2, viene perciò analizzata nel dettaglio:

- **Parametri in ingresso**:
  - `input`, un puntatore ad un array di `int32` contenente i campioni della serie temporale.
  - `samplecount`, un intero che indica il numero di campioni dati in ingresso alla funzione.
  - `diff0`, il valore della differenza rispetto al pacchetto precedente.
  - `swapflag`, un valore di configurazione.
- **Output**
  - `output`, il puntatore all'array in cui verranno memorizzati i valori restituiti dalla funzione di codifica STEIM2.
  - `outputlenght`, indica quanto sarà lungo l'array contenente i valori di output.

– `byteswritten`, indica la lunghezza in byte della serie di valori compressi.

Capitolo 4

Analisi dei risultati

4.1 Tempistiche di serializzazione per ogni formato

Il primo test svolto è stata la stima del tempo di serializzazione per ogni formato di pacchettizzazione.

Si prenda in considerazione il seguente pacchetto per comprendere a pieno i dati che sono stati serializzati:

```
{
    "i": "IV_NRCA_00_BHE_D",
    "t": "2023-04-23T00:00:00",
    "e": 10,
    "v_x": [1,0,3,-4,2...],
    "v_y": [3,2,-1,-3,1...],
    "v_z": [1,0,3,-4,2...],
}
```

4.2 Test su tempistiche di serializzazione

4.2.1 MiniSEED con codifica INT32

Vengono ora graficate [1] le tempistiche di serializzazione in funzione del numero di campioni inseriti in ciascun pacchetto. In questo caso si fa riferimento al formato MiniSEED con codifica INT32, quindi per ogni campione della forma d'onda vengono allocati `4Bytes`.

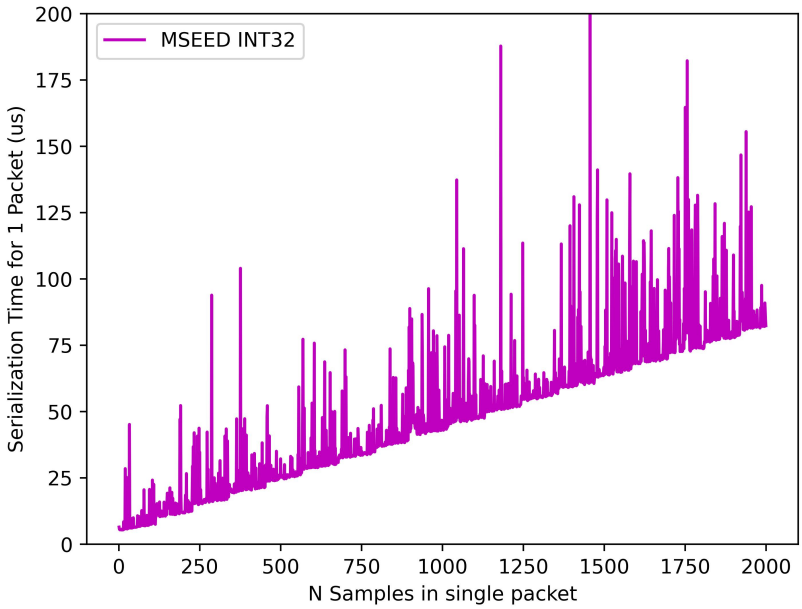

Figura 4.1: Tempistiche di serializzazione in funzione del numero di campioni di un pacchetto MiniSEED con codifica INT32

Dal grafico è possibile evidenziare alcune caratteristiche:

- Le tempistiche di serializzazione non scendono mai sotto una certa soglia, indice che quella è la soglia minima per l'esecuzione delle istruzioni macchina necessarie alla serializzazione e per l'allocazione della memoria.
- Come atteso, i tempi variano linearmente con la dimensione dei pacchetti.

[1] Trasformare i risultati in grafici.

- La misura è molto "rumorosa". Purtroppo, effettuando test del genere su sistemi operativi non realtime (MacOS in questo caso) non è possibile intervenire finemente sulle decisioni prese dallo schedulatore.

In generale, quando si parla di early warning, si tengono in considerazione misure dalla durata di circa 1s.

Nel nostro caso, il campionamento viene effettuato a $100Hz$, pertanto un pacchetto contenente 1 secondo di forma d'onda contiene 100 campioni e richiede circa $10\mu s$ per essere elaborato.

4.2.2 MiniSEED con codifica STEIM2

In Figura 4.2 si riporta il grafico ottenuto serializzando pacchetti MiniSEED con codifica STEIM2.

A differenza della situazione precedente, in cui veniva usata una codifica INT32, qui i pacchetti - a parità di dimensioni - contengono un numero variabile di informazioni. Tale quantità dipende dal fattore di compressione che la codifica STEIM2 è stata in grado di introdurre.

4.2.3 Protobuf

Per quanto riguarda la serializzazione con protobuf si hanno risultati paragonabili ai precedenti. Anche in questo caso la misura risulta essere molto rumorosa.

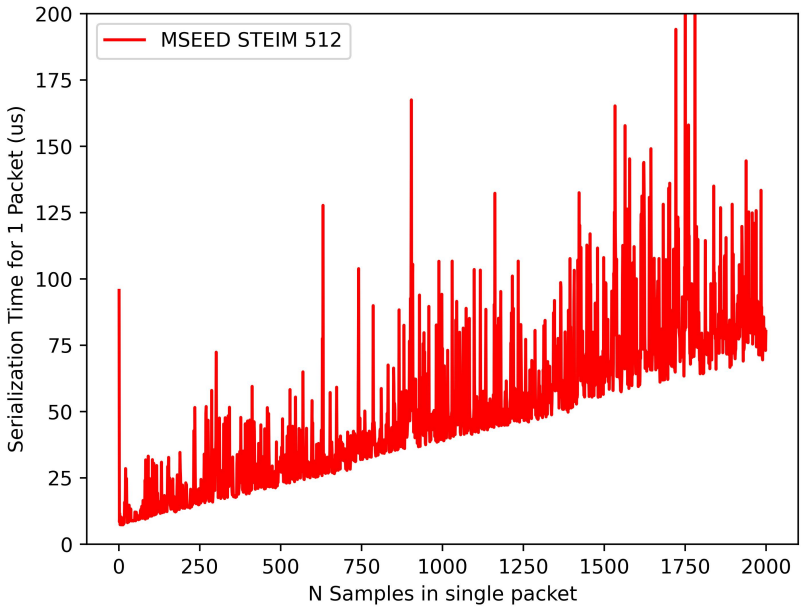

Figura 4.2: Tempistiche di serializzazione in funzione del numero di campioni di un pacchetto MiniSEED con codifica STEIM2

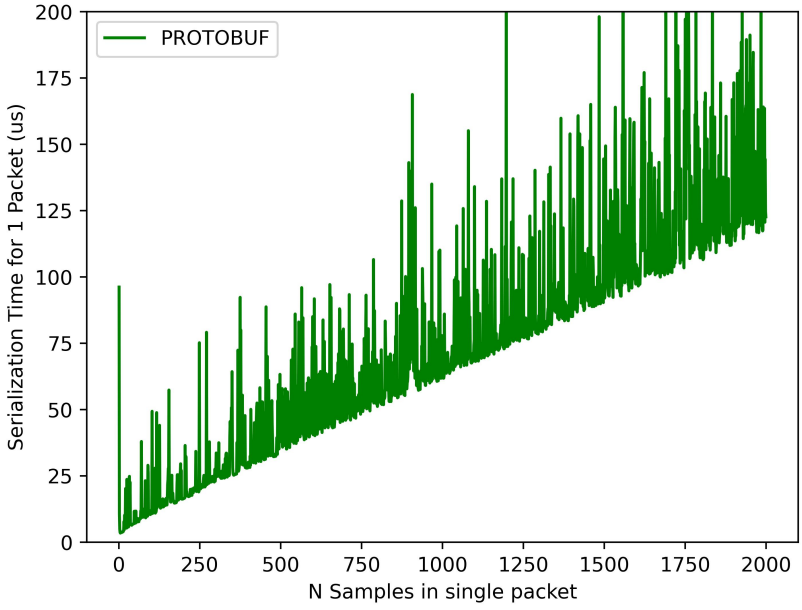

Figura 4.3: Tempistiche di serializzazione in funzione del numero di campioni di un pacchetto Protobuf

Si precisa che nella generazione del file `*.proto`, in cui viene definita la struttura dell'oggetto da serializzare, sono state applicate alcune accortezze:

- Il tipo degli elementi dell'array contenenti le forme d'onda è stato configurato come `sint32`. Questo tipo ci permette sia di effettuare un confronto alla pari con gli altri formati dato che si tratta sempre di un valore a `4Bytes`, sia di ottimizzare le performance del codificatore protobuf.
- È stata utilizzata la word `packed` per ottimizzare la codifica dell'array. Tale word potrebbe anche essere omessa, in quanto dalla versione 3 di protobuf è un opzione abilitata di default.

4.2.4 JSON

Viene effettuato un test anche con JSON, ma è possibile visionare nei log tempistiche molto alte già da quando si serializzano pacchetti piccoli. Sembra non essere una scelta ottimale.

4.2.5 CBOR

L'ultimo formato preso in considerazione è il CBOR. Esso è nato per essere veloce e conciso, pertanto ci si aspettano ottime performance da esso.

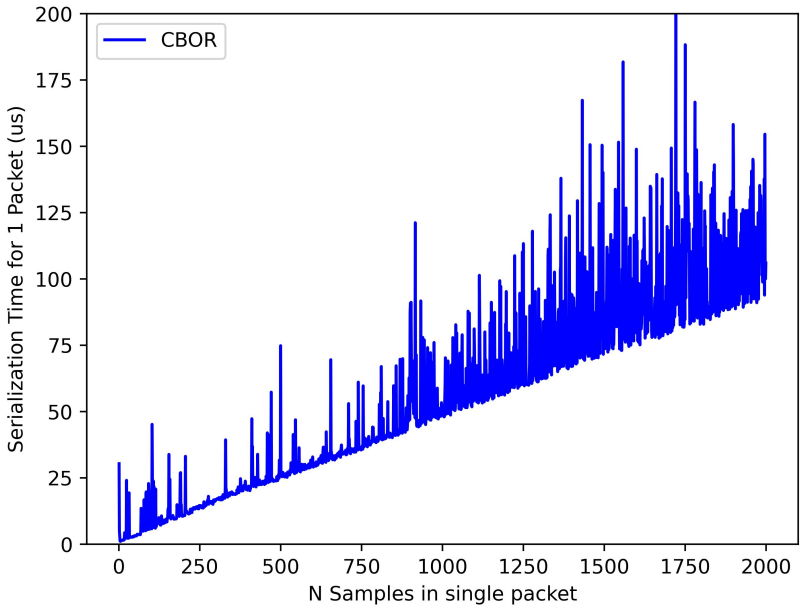

Figura 4.4: Tempistiche di serializzazione in funzione del numero di campioni di un pacchetto CBOR

Ancora una volta la misura risulta essere piuttosto rumorosa, nonostante ciò è chiaramente visibile un andamento di massima.

Si tiene a precisare che nella generazione del pacchetto CBOR non sono state inserite delle mappe di oggetti, pertanto per decodificare i valori giusti è assolutamente necessario seguire lo stesso ordine che si è seguito in fase di codifica.

La scelta di non inserire mappe è stata fatta in modo da ottimizzare il più possibile il pacchetto.

4.2.6 Comparazione dei formati

Si pongono ora tutti i formati nello stesso grafico allo scopo di valutarli e individuarne un migliore:

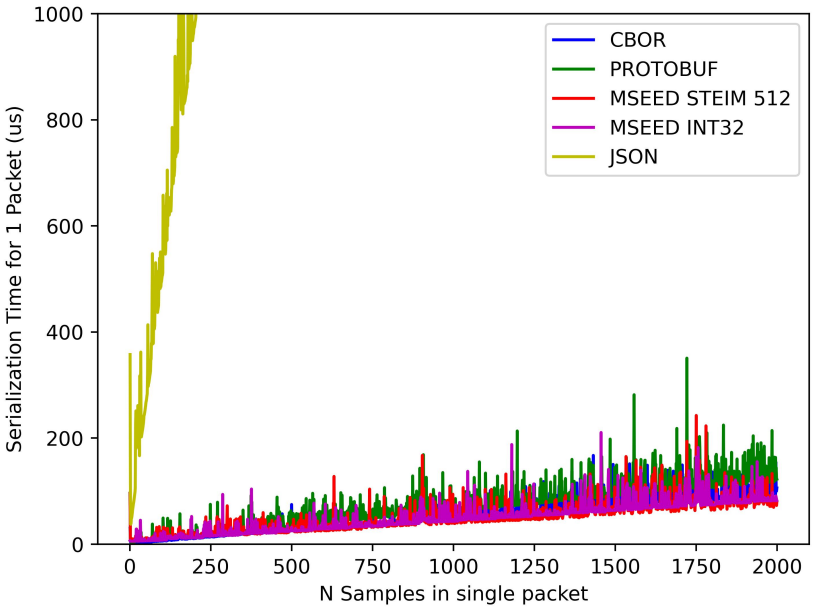

Figura 4.5: Comparazione di tutti i formati di serializzazione utilizzati

Dal grafico in Figura 4.5 è impossibile non notare le performance estremamente scarse del formato **JSON**. Tale formato, nonostante sia ottimo in moltissime situazioni essendo umanamente comprensibile, non è utilizzabile quando si vogliono avere performance ottimali in termini di velocità di serializzazione. Questa scarsa velocità è dovuta al fatto che ogni Byte deve essere codificato secondo lo standard UTF-8.

Viene quindi riproposto, in Figura 4.6, lo stesso grafico, questa volta con una scala più adeguata e rimuovendo il formato JSON.

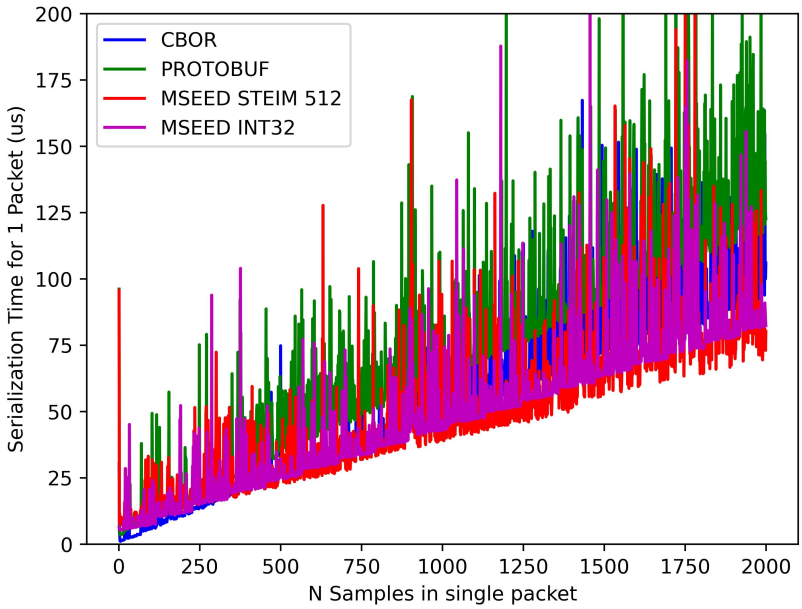

Figura 4.6: Comparazione di tutti i formati di serializzazione utilizzati senza considerare il JSON

A causa dell'elevata rumorosità delle misure, il grafico risulta quasi illeggibile.
L'unica informazione che si può ricavare è che qualsiasi formato non scende mai sotto una certa soglia.

4.3 Calcolo delle tempistiche medie

Per tentare di ridurre il rumore nelle misure si è ben pensato di calcolare le tempistiche **medie** su un insieme di 5000 pacchetti identici.

Pertanto il software di benchmarking[2] è stato modificato per generare 5000 campioni identici ad ogni ciclo (anzichè 1), e misurarne il tempo totale di serializzazione.
Il tempo totale è infine stato diviso per il numero di campioni (5000), così da ottenerne il valore medio.

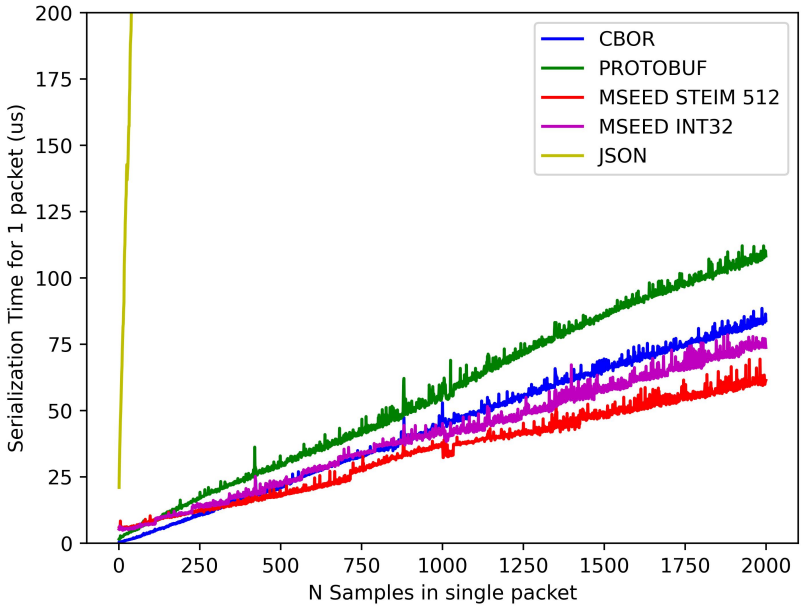

Figura 4.7: Comparazione dei tempi medi di tutti i formati di serializzazione utilizzati

Come mostrato in Figura 4.7, ora gli andamenti sono molto più lineari e si potrenne pensare di approssimarli con delle rette.

[2]strumenti progettati per valutare le prestazioni e le capacità di un sistema informatico o di specifici componenti hardware o software. Questi programmi eseguono test mirati per misurare e confrontare le prestazioni di diverse configurazioni

Inoltre, il JSON, si conferma estremamente più lento rispetto agli altri formati.

Per una migliore analisi viene riproposto lo stesso grafico in Figura 4.8, questa volta eliminando il JSON.

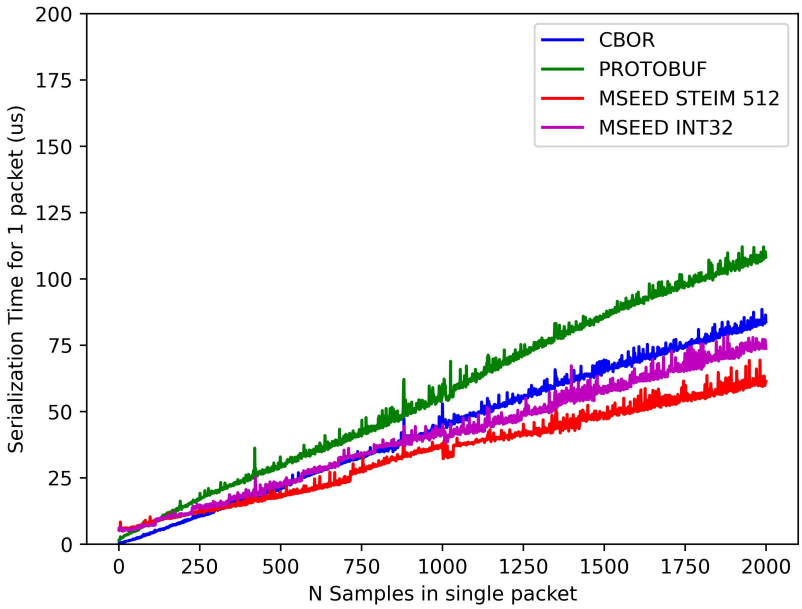

Figura 4.8: Comparazione dei tempi medi di tutti i formati di serializzazione utilizzati escludendo il JSON

Analisi dei dati ottenuti

La funzione `polyfit` di Python, linguaggio con cui sono stati analizzati i dati, ci permette di generare il polinomio di grado arbitrario che minimizza l'errore medio di una funzione. In altri termini, scegliendo il grado del polinomio pari a 1, si possono ottenere le linee di tendenza.

Viene quindi applicata la seguente funzione ad ogni andamento del grafico precedente (eccetto al JSON).

```
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
```

È quindi possibile apprezzare il grafico ottenuto facendo riferimento alla Figura 4.9

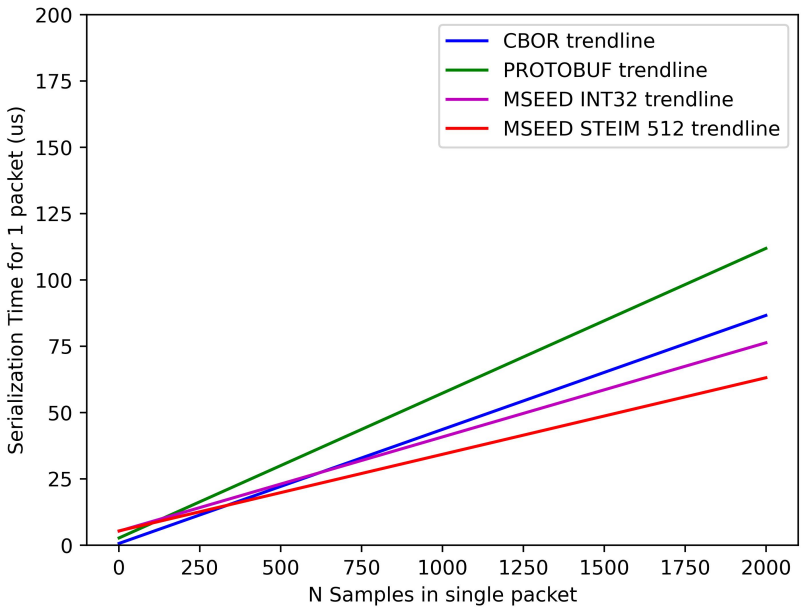

Figura 4.9: Comparazione delle trendlines dei tempi medi di tutti i formati di serializzazione utilizzati, escludendo il JSON

A seguito di questa breve analisi sembra che per pacchetti grandi il formato più veloce a serializzare sia il MiniSEED con codifica STEIM2, mentre per pacchetti piccoli sembra che le migliori performance si abbiano con il CBOR.
In Figura 4.10 viene proposto un grafico con soltanto i due formati più interessanti.

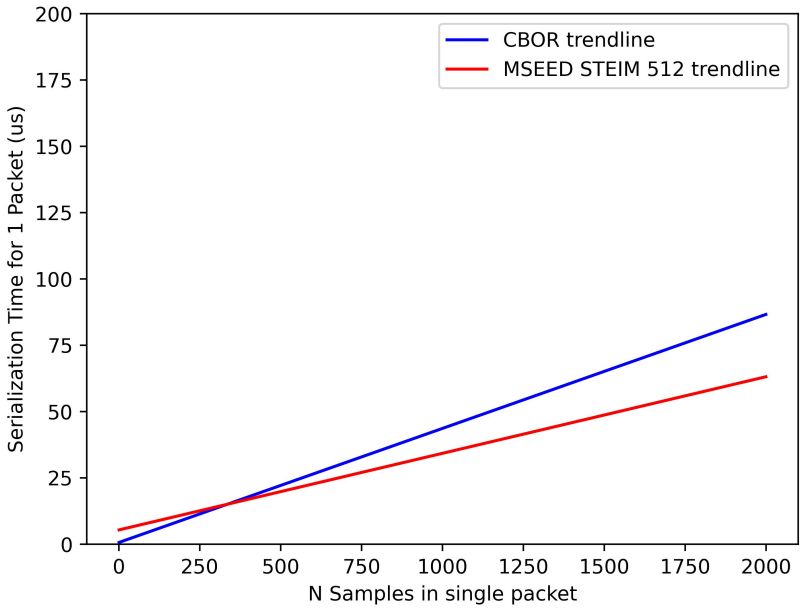

Figura 4.10: Comparazione delle trendlines di MiniSEED STEIM2 e CBOR

Possiamo fare alcune considerazioni su quanto ottenuto:

- In sistemi di early warning si ha la necessità di trasferire serie temporali brevi, generalmente della durata inferiore ai 2 secondi. Sebbene il MiniSEED con STEIM2 offra un buon fattore di compressione per pacchetti grandi, non risulta ottimale in pacchetti piccoli.

Considerazioni sulle tempistiche di serializzazione

A seguito dei numerosi test effettuati sono emersi alcuni andamenti particolari che vengono di seguito riassunti:

- **Linearità con la quantità di informazione contenuta**, i grafici mostrati hanno tutti un andamento piuttosto lineare in funzione della quantità di dati inserita in ogni pacchetto.
- **Rapidità di serializzazione**, nonostante le numerose differenze trovate tra i vari formati, tutti questi permettono di avere tempistiche di serializ-

zazione estremamente basse. In pacchetti piccoli, con quantità di campioni adatte a sistemi di early warning, nel peggiore dei casi, non si superano i $20\mu s$.

4.4 Dimensione dei pacchetti

In sistemi di streaming real time dei dati è di fondamentale importanza trasferire solo i dati necessari. Ogni dato superfluo o ridondante comporta inutile perdita di tempo e spreco di risorse.

Vengono pertanto studiate le dimensioni dei pacchetti generate dai diversi formati di serializzazione.

Si prenda in considerazione il seguente pacchetto per comprendere a pieno i dati su cui si eseguiranno i test:

```
{
    "i": "IV_NRCA_00_BHE_D",
    "t": "2023-04-23T00:00:00",
    "e": 10,
    "v_x": [1,0,3,-4,2...],
    "v_y": [3,2,-1,-3,1...],
    "v_z": [1,0,3,-4,2...],
}
```

4.4.1 MiniSEED INT32

Facendo riferimento allo schema di informazioni che dobbiamo inserire nel pacchetto e facendo riferimento alla struttura del MiniSEED, possiamo calcolare in maniera piuttosto semplice e deterministica la dimensione di un pacchetto MiniSEED con codifica INT32.
Si ricorda, inoltre, che qualora si vogliano tenere in considerazione 3 assi - come nel nostro caso - sarà necessario generare 3 pacchetti.

Il dato finale avrà perciò le seguenti dimensioni:

$$SIZE_{totale} = SIZE_{pacchetto} \times 3$$

In cui la dimensione di ogni pacchetto viene calcolata sommando la dimensione dell'header, la dimensione dei due blocchetti dello standard miniSEED e la dimensione degli N campioni della serie temporale. In questa codifica la dimensione di ogni campione è di `4Bytes`:

$$SIZE_{pacchetto} = 48 + 16 + (N \times 4)$$

Pertanto, se si vogliono serializzare un numero N di campioni pari a 200 per ogni asse, la dimensione complessiva dei pacchetti MiniSEED è di `2592Bytes`.

4.4.2 MiniSEED STEIM2

Nel caso in cui venga applicata la codifica STEIM2 al formato di serializzazione MiniSEED, si avranno, presumibilmente, dimensioni complessive minori dato che la codifica STEIM2 effettua una compressione del dato.

Si possono fare alcune considerazioni riguardo questo genere di codifica:

- Ciascun pacchetto contiene un header più i due blocchetti, per un totale di `64Bytes` ciascuno.
- Ci si aspetta una dimensione complessiva inferiore rispetto a quella che si ha con la codifica INT32.
- Quando un pacchetto supera i `512Bytes` viene aggiunto un nuovo pacchetto, quindi, un nuovo header.

Si precisa che la codifica STEIM2 alloca dei frame di `8Bytes` alla volta, pertanto, l'andamento atteso dal grafico che mette in relazione quantità di campioni inseriti in un singolo pacchetto e dimensione del pacchetto, avrà un andamento a gradini.

4.4.3 CBOR

Serializzando un pacchetto in CBOR, la sua dimensione sarà dettata dai byte di controllo, nonché dalla dimensione delle informazioni contenute in un pacchetto.

Si fa sempre riferimento al seguente pacchetto:

```
{
    "i": "IV_NRCA_00_BHE_D",
    "t": "2023-04-23T00:00:00",
    "e": 10,
    "v_x": [1,0,3,-4,2...],
    "v_y": [3,2,-1,-3,1...],
    "v_z": [1,0,3,-4,2...],
}
```

Serializzandolo in CBOR si ottiene la seguente serie di bytes:

```
86                                      # array(6)
   18 2a                                #   unsigned(42)
   73                                   #   text(19)
      323031322d30312d30315430303a30303a3030 #     "2023-04-23T00:00:00"
   70                                   #   text(16)
      49565f4e5243415f30305f4248455f44   #     "IV_NRCA_00_BHE_D"
   8a                                   #   array(10)
      03                                #     unsigned(3)
      01                                #     unsigned(1)
      00                                #     unsigned(0)
      01                                #     unsigned(1)
      20                                #     negative(-1)
      01                                #     unsigned(1)
      20                                #     negative(-1)
      22                                #     negative(-3)
      21                                #     negative(-2)
      26                                #     negative(-7)
   8a                                   #   array(10)
      03                                #     unsigned(3)
      01                                #     unsigned(1)
      00                                #     unsigned(0)
      01                                #     unsigned(1)
      20                                #     negative(-1)
      01                                #     unsigned(1)
      20                                #     negative(-1)
      22                                #     negative(-3)
      21                                #     negative(-2)
      26                                #     negative(-7)
   8a                                   #   array(10)
```

```
03                                      #     unsigned(3)
01                                      #     unsigned(1)
00                                      #     unsigned(0)
01                                      #     unsigned(1)
20                                      #     negative(-1)
01                                      #     unsigned(1)
20                                      #     negative(-1)
22                                      #     negative(-3)
21                                      #     negative(-2)
26                                      #     negative(-7)
```

I bytes presenti nel pacchetto sono così distribuiti:

- `1 Byte` determina il numero di dati contenuti nella struttura principale, in questo caso 6.
- `2 Bytes` determinano il parametro intero che serve ad indicare la codifica utilizzata "e".
- `1 Byte` determina il valore effettivo della codifica utilizzata.
- `1 Byte` determina la presenza di una stringa di testo contenente un timestamp.
- `19 Bytes` rappresentano il valore del timestamp.
- `1 Byte` determina la presenza di una stringa di testo contenente i metadati del sismografo.
- `16 Bytes` rappresentano il valore del metadato.
- `3 Bytes` servono per dichiarare i 3 array contenenti le forme d'onda.
- Gli ulteriori Bytes rappresentano i valori delle serie temporali.

In questo caso non è immediato definire con formule matematiche la quantità di Byte necessari per rappresentare le serie temporali, in quanto CBOR utilizza

una variable-lenght encoding. Pertanto la dimensione di un campione dipende dal valore contenuto in esso.

4.4.4 JSON

In maniera analoga al CBOR, la dimensione di un pacchetto JSON sarà determinata da una quantità fissa di metadati ("e", "t", "i), più una parte che dipenderà soltanto dalla lunghezza della serie temporale.

Riguardo la dimensione complessiva di un pacchetto JSON è decisamente impattante l'ingente quantità di simboli necessari alla codifica, come parentesi, virgole e apici. Nonostante ciò, anche in JSON, è possibile sfruttare l'efficacia di una variable-leenght encoding per gli interi contenuti nella serie temporale. Pertanto ci si aspettano performance complessive migliori di quelle che si hanno nel formato MiniSEED INT32.

```
{
    "i": "IV_NRCA_00_BHE_D",
    "t": "2023-04-23T00:00:00",
    "e": 10,
    "v_x": [1,0,3,-4,2...],
    "v_y": [3,2,-1,-3,1...],
    "v_z": [1,0,3,-4,2...],
}
```

4.4.5 ProtoBuf

In maniera del tutto similare a quanto visto con JSON e CBOR, il Protobuf permette di adottare la variable-lenght encoding. In questo modo siamo in grado di sfruttare la proprietà delle serie temporali di avere generalmente campioni con valori numerici bassi (al di fuori dell'evento sismico).

4.4.6 Confronto delle dimensioni dei pacchetti

Nel seguente grafico vengono messi a confronto tutti gli algoritmi di serializzazione considerati fino ad ora. Il grafico in Figura 4.11 rappresenta l'andamento della dimensione dei pacchetti generati in funzione della quantità di campioni della serie temporale inseriti in ciascun pacchetto.

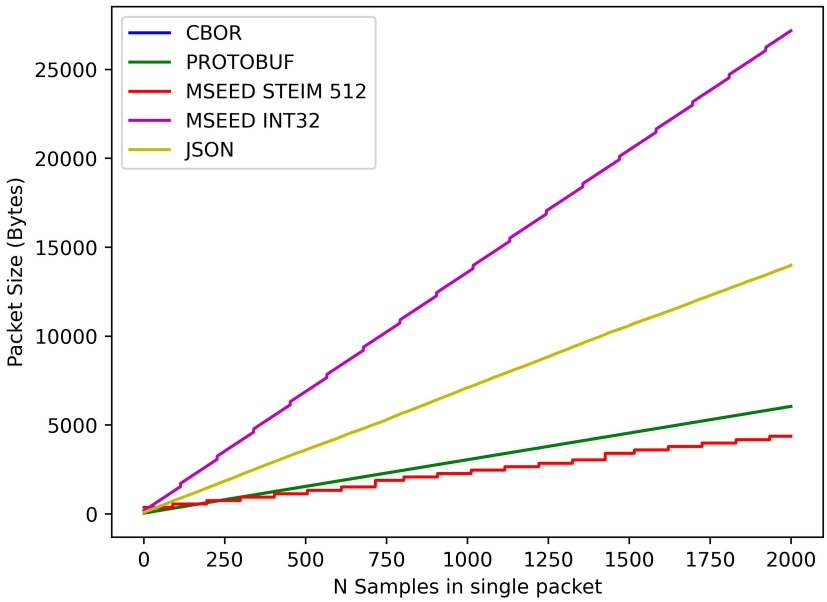

Figura 4.11: Andamento delle dimensioni dei pacchetti in funzione dei campioni inseriti in ciascun pacchetto

Come atteso. tutti i formati hanno un andamento con tendenza lineare.

Considerazioni su MiniSEED INT32

Il formato MiniSEED INT 32, come chiaramente osservabile in Figura 4.11, ha l'andamento peggiore.

L'elevata dimensione di ogni pacchetto è dovuta principalmente dal fatto che

non viene utilizzata nessun tipo compressione, ne di variable-lenght encoding.

Si ponga ora l'occhio su un'altra indistinguibile caratteristica: L'andamento a gradini. Di tanto in tanto, nell' andamento MiniSEED INT32, sono osservabili dei piccoli salti.
Tali salti sono causati dal fatto che MiniSEED INT32, nel momento in cui un pacchetto supera i `512 Bytes`, lo chiude e crea un nuovo pacchetto (3 se consideriamo di serializzare 3 assi), aggiungendo un nuovo header.

Considerazioni su JSON

L'andamento del formato JSON non sembra avere numerose criticità, nonostante ciò, introducendo numerosi simboli per la codifica, risulta essere meno efficiente degli altri, sia per quanto riguarda la velocità di serializzazione che per quanto riguarda la dimensione dei pacchetti generati.

Considerazioni su CBOR e Protobuf

Entrambi i formati permettono di raggiungere ottime performance per quanto riguarda le dimensioni dei pacchetti codificati.
Si tende in ogni caso a preferire il CBOR, in quanto più veloce e più semplice da implementare. Le librerie necessarie per gestire il Protobuf sono di gran lunga più complesse di quelle necessarie all'utilizzo del CBOR.

Considerazioni su MiniSEED STEIM2

Applicando l'algoritmo di compressione STEIM2 al formato MiniSEED le performance che si ottengono sembrano essere davvero notevoli.
Per serie temporali dalla lunghezza inferiore ai 300 campioni si preferisce comunque il CBOR.

A causa dell' header introdotto con ogni pacchetto al superamento dei `512 Bytes`, il MiniSEED STEIM2 riesce ad ottenere risultati migliori solo per pacchetti grandi. Questo è dovuto al fatto che in pacchetti grandi, l'ingente fattore di compressione che fornisce STEIM2, va ad ammortizzare lo spreco di memoria causato dagli header ridondanti.

Si faccia ora attenzione all'andamento a gradini del MiniSEED con STEIM2. Tale andamento è principalmente determinato dalla codifica STEIM2 che alloca frame da `8Bytes` alla volta.

4.5 Variazione della dimensione dei pacchetti

Fino ad ora sono state valutate le dimensioni dei pacchetti contenenti un tratto generico di una serie temporale.
È quindi auspicabile chiedersi se ci siano delle variazioni nella dimensione dei pacchetti in punti particolari della serie temporale, come ad esempio nel momento in cui si manifesta l'evento sismico.

4.5.1 Test con finestra scorrevole

Per determinare le variazioni nelle dimensioni è stato quindi generato un pacchetto contenente un numero fisso di campioni per ogni tratto della serie temporale.

In altre parole è stata utilizzata una finestra scorrevole con numero fisso di campioni. Sono stati effettuati 4 test, ciascuno con la dimensione della finestra fissata ad un certo valore: 100, 200, 300 e 500.

Vengono di seguito proposte delle coppie di grafici.
Il grafico nella parte superiore delle figure rappresenta l'andamento della dimensione del pacchetto in funzione dello slot della serie temporale a cui si fa riferimento.
Il grafico inferiore rappresenta l'intera serie temporale.

Si tiene a precisare che più il grafico è basso, minore è la dimensione dei pacchetti, pertanto si hanno performance migliori.

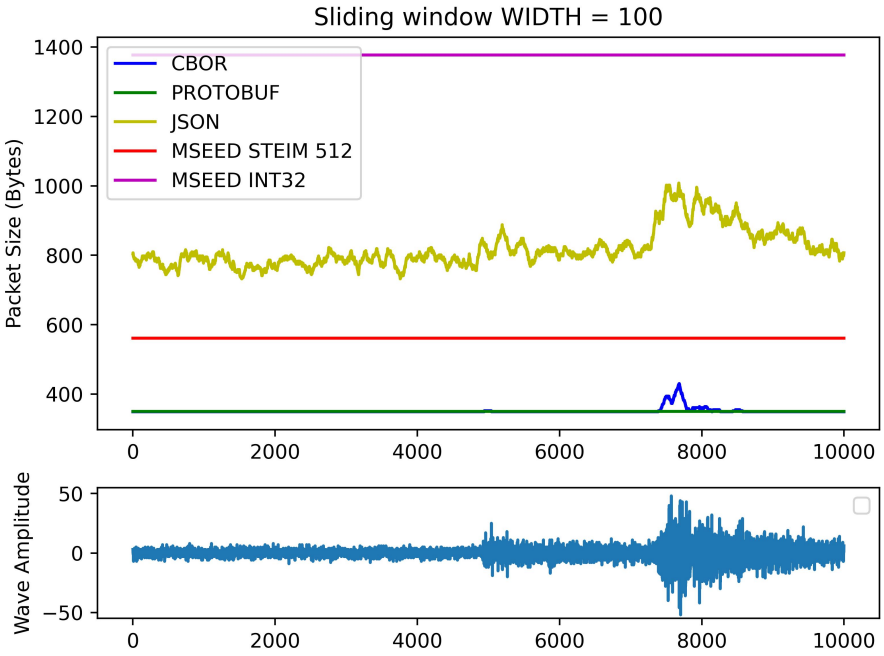

Figura 4.12: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 100 campioni

Anche da questo primo esperimento si notano numerose caratteristiche che contraddistinguono ciascun formato di serializzazione.

Prima di trarre delle conclusioni viene riproposto lo stesso esperimento, ma considerando finestre scorrevoli di diversa dimensione.

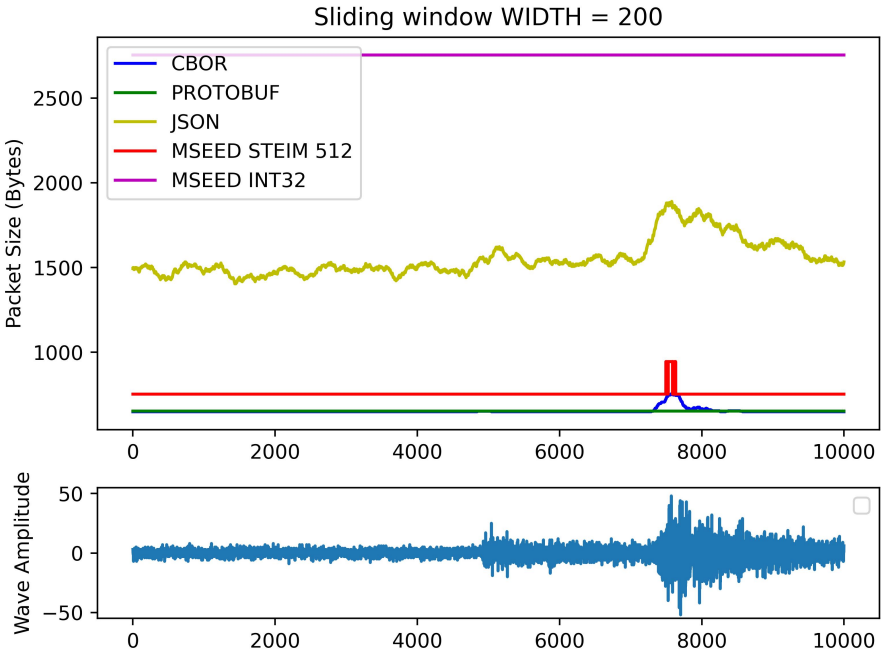

Figura 4.13: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 200 campioni

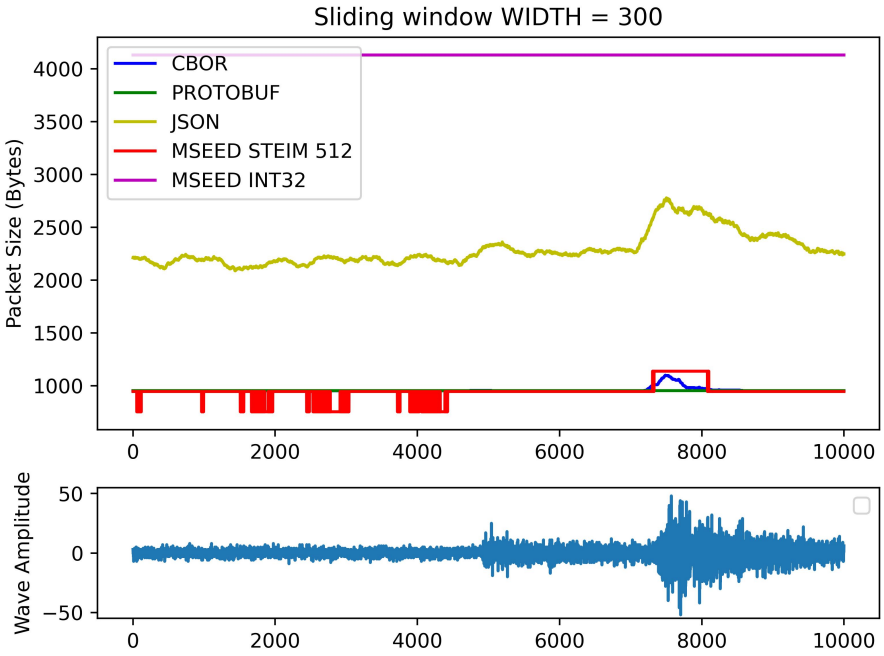

Figura 4.14: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 300 campioni

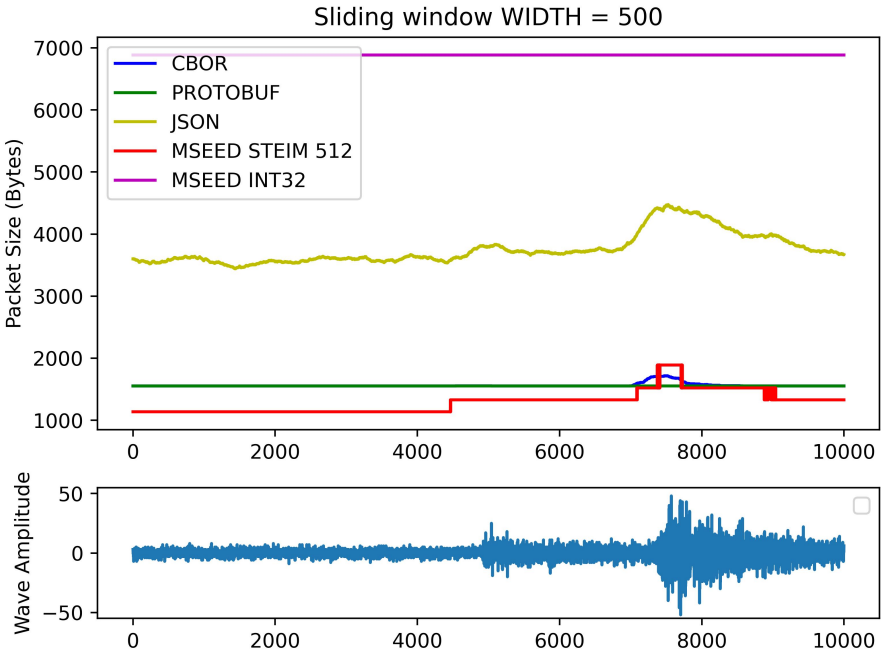

Figura 4.15: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 500 campioni

La rappresentazione più interessante è quella in Figura 4.15, essa, infatti, mette in risalto in maniera piuttosto evidente le caratteristiche dei vari formati di serializzazione.

MiniSEED INT32

Il formato MiniSEED con codifica INT32, nel seguente test, mostra un andamento completamente piatto.
Questo particolare andamento è dovuto dal fatto che tale formato non utilizza alcuna variable-lenght encoding. Pertanto la dimensione finale di un pacchetto dipende soltanto dalla quantità dei campioni inseriti in esso (quantità fissa in questo test), e non dal loro valore.

JSON

In JSON si ha un grafico piuttosto oscillante. Tale grafico, però, segue l'andamento della serie temporale.
Questo andamento è determinato dal fatto che il valore dei campioni della serie temporale vengono codificati come stringa. Pertanto il contributo che ogni campione da alla dimensione complessiva del pacchetto dipende soltanto dalla quantità di cifre con cui viene rappresentato il valore (ed eventualmente dal segno `"-"`).

Nel punto in cui si manifesta l'evento sismico avremo quindi valori maggiori della serie temporale, e di conseguenza dimensione del pacchetto complessivamente maggiore.

CBOR

Il CBOR utilizza una variable-lenght encoding, pertanto è in grado di fornire performance complessivamente migliori.

Nonostante ciò, nel momento in cui si inseriscono valori grandi, questi hanno bisogno di più Bytes per essere rappresentati, il che porta ad un inevitabile incremento complessivo nelle dimensioni del pacchetto nel momento in cui si manifesta il fenomeno sismico.

MiniSEED STEIM2

Come atteso, il formato MiniSEED con codifica STEIM2 ha un andamento a gradini. Tale andamento è dovuto al fatto che STEIM2 alloca dei frame di `8 Bytes` alla volta riempiendoli solo quando necessario.

Nel momento in cui si manifesta l'evento sismico, la serie temporale conterrà valori con con più variabilità, con conseguente riduzione del rapporto di

compressione. Questo porta indubbiamente all'allocazione di più frame per contenere il dato.

4.6 Considerazioni su tempistiche e dimensioni del pacchetto

A seguito dell'analisi dei risultati ottenuti si può concludere che le tempistiche di serializzazione sono profondamente influenzate dal tempo richiesto per scrivere il dato in ram. Aumentare la complessità dell'algoritmo introducendo un sistema di compressione sembra portare vantaggi in termini di performance complessive. In virtù di questo è stato scelto di applicare la compressione STEIM2 al formato di serializzazione più semplice,leggero e veloce: CBOR.

4.7 Soluzione proposta

A seguito di un'approfondita analisi dei dati ottenuti dagli esperimenti dei capitoli precedenti, si è giunti alla conclusione che le tempistiche di serializzazione non dipendono soltanto dal numero di operazioni necessario a serializzare i pacchetti, dipendono anche dalla dimensione dei pacchetti stessi.
È cruciale generare pacchetti piccoli in quanto le operazioni di salvataggio dei dati in RAM, a volte, potrebbero essere un bottleneck per le tempistiche.
Ma non solo, generare pacchetti piccoli è sicuramente ottimale anche per migliorare le tempistiche di trasmissione dei dati in rete.

L'esempio più lampante si ha confrontando il Formato MiniSEED con codifica INT32 e lo stesso formato ma con codifica STEIM2.
Essendo STEIM2 un algoritmo di compressione differenziale è chiaro che esso richieda l'esecuzione di numerosi istruzioni in più rispetto al banale salvataggio dei dati in un array INT32. Questo potrebbe portare a pensare che il MiniSEED con codifica INT32 sia più veloce del MiniSEED con codifica STEIM2, ma ciò viene smentito dai dati ottenuti in fase di test.

A quanto pare le tempistiche richieste per salvare i dati in RAM non sono trascurabili. Questo porta ad un ritardo dipendente dalla dimensione del pacchetto non sotto-valutabile.

A seguito di queste considerazioni si è pensato di integrare la codifica STEIM2 all'interno di un pacchetto CBOR.

In linea teorica la congiunzione del CBOR con questo formato di compressione dovrebbe portare a numerosi benefici, quali:

- Riduzione delle dimensioni complessive del pacchetto grazie alla compressione STEIM2.
- Maggior velocità nella serializzazione permessa dalla compressione STEIM2 che riducendo le dimensioni del pacchetto ne agevola la sua scrittura in RAM.

- Possibilità di inserire nel pacchetto un numero arbitrario di campioni. Non si ha più la necessità di aggiungere headers ridondanti.

4.7.1 Dimensioni del pacchetto

Applicando la codifica di compressione STEIM la quantità di Bytes contenuti nel pacchetto si riduce.

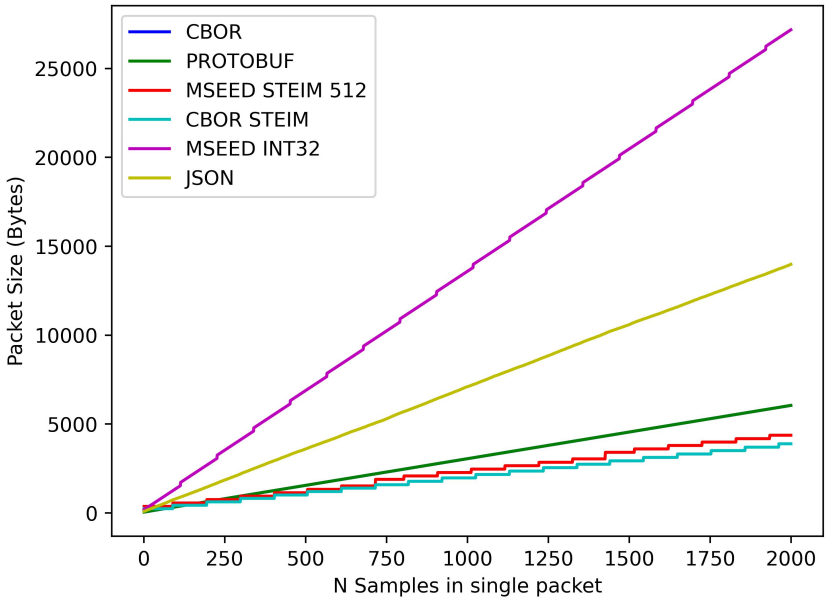

Figura 4.16: Variazione della dimensione del pacchetto in funzione della quantità di campioni inseriti in ciascun pacchetto.

Come si vede dalla Figura 4.16 l'andamento generato dal formato proposto è migliore, seppur di poco, rispetto tutte le soluzioni classiche. Si può inoltre notare il tipico andamento a gradini dovuto alla compressione STEIM2 che alloca un frame di 8Byte alla volta.

Consideriamo ora lo stesso grafico di Figura 4.16, questa volta includendo soltanto MiniSEED STEIM2 e le due versioni CBOR. Il grafico viene adattato

ad un'altra scala per migliorarne la leggibilità.

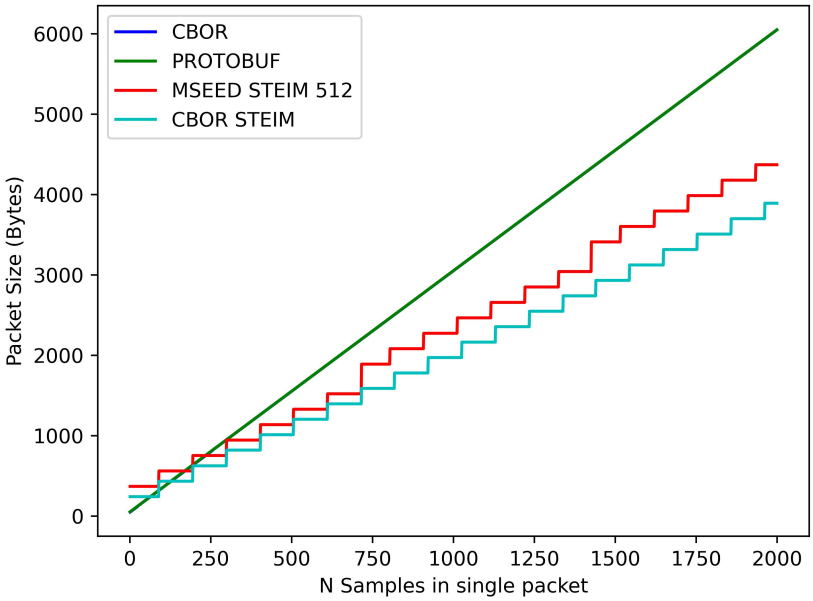

Figura 4.17: Variazione della dimensione del pacchetto in funzione della quantità di campioni inseriti in ciascun pacchetto, nel dettaglio.

Ora che la differenza è apprezzabile si può chiaramente osservare come il formato proposto sia migliore rispetto a quelli standard circa in ogni situazione.

Si può inoltre vedere la sempre maggiore discrepanza con il MiniSEED STEIM man mano che si aumenta la quantità di campioni.
Questo incremento di discrepanza tra MiniSEED STEIM2 e CBOR STEIM2 è dovuta dal fatto che quando un pacchetto MiniSEED supera i `512Bytes`, viene generato un nuovo pacchetto, con la conseguente aggiunta di un ulteriore header.
Ricordiamo che si stanno serializzando campioni ottenuti da accelerometri a 3 assi, pertanto per ogni asse è necessario generare un pacchetto miniSEED, con la conseguente aggiunta di 3 headers. Utilizzando la soluzione proposta, è

possibile utilizzare un solo header qualsiasi sia la dimensione del pacchetto.

4.7.2 Variazione della dimensione dei pacchetti

Viene ora riproposto il test della finestra scorrevole. Viene considerato un numero fisso di campioni e si analizza in che modo cambia la dimensione complessiva del pacchetto in funzione del punto della serie temporale in cui si considera lo slot.

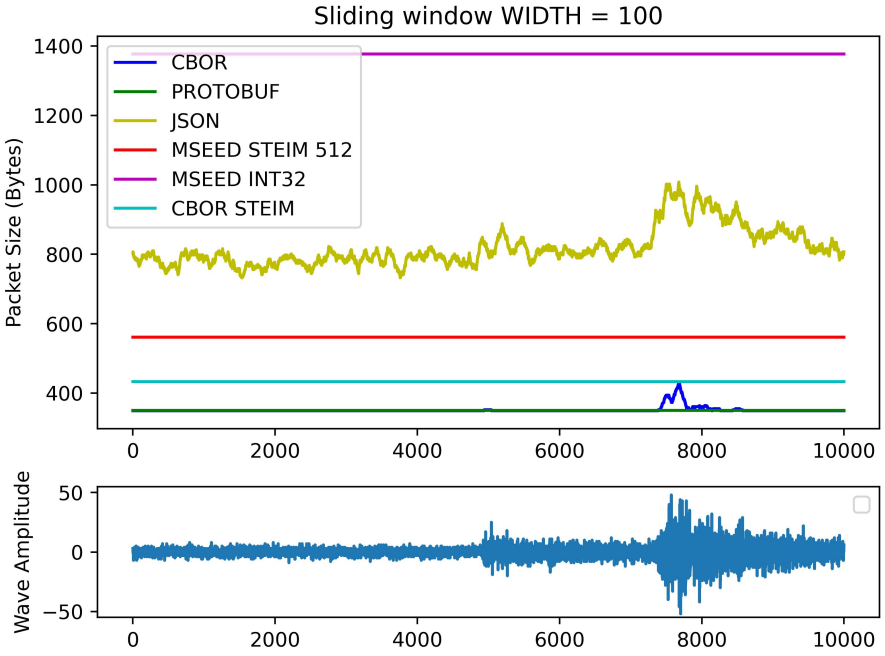

Figura 4.18: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 100 campioni

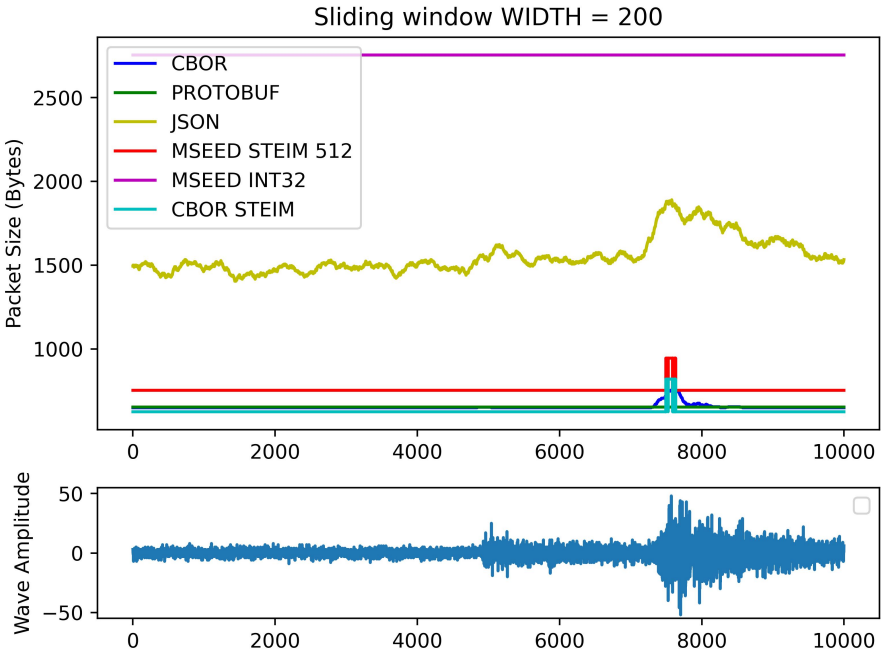

Figura 4.19: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 200 campioni

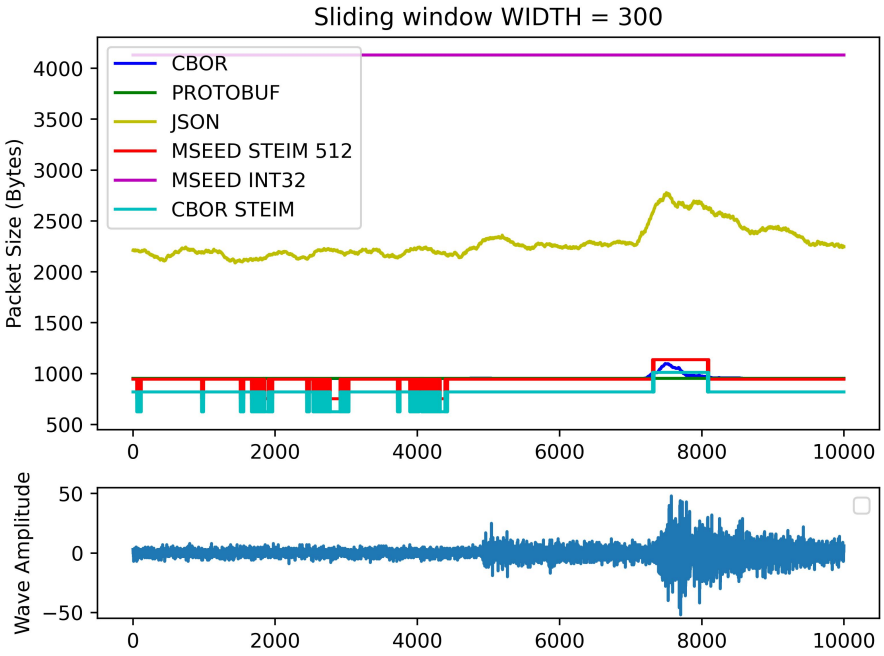

Figura 4.20: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 300 campioni

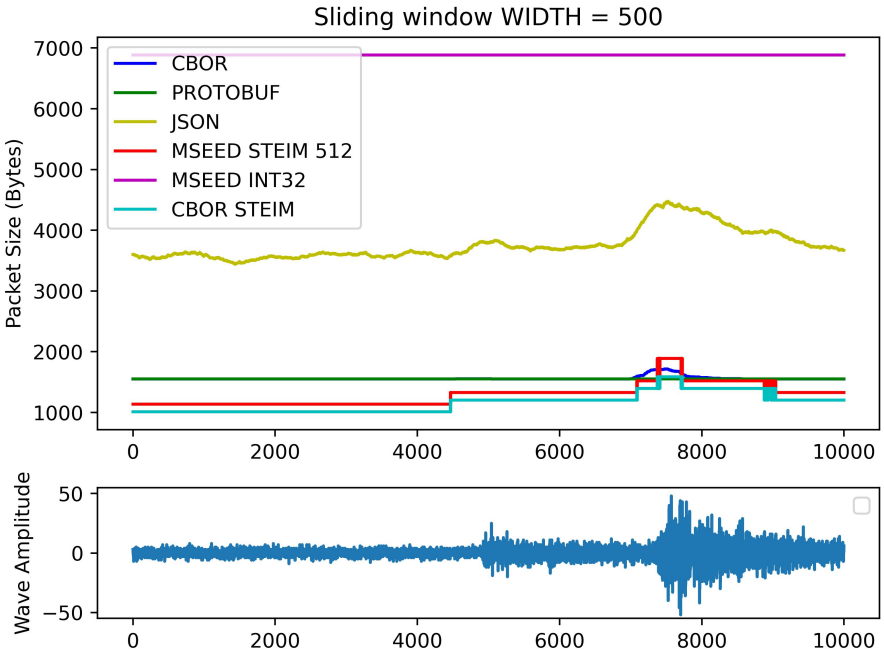

Figura 4.21: Variazione della dimensione del pacchetto in funzione dello slot della serie temporale che viene considerato. Finestra da 500 campioni

Si ricava dai grafici che le performance migliorano anche nel caso in cui ci sia un'ingente variazione nella serie temporale. Questo permette di affermare che, nel momento in cui siano di interesse soltanto le dimensioni del pacchetto, non c'è situazione in cui non sia conveniente utilizzare il formato proposto.

4.7.3 Tempistiche di serializzazione

Vengono infine determinate sperimentalmente le tempistiche di serializzazione del formato proposto. Ciò che ci si aspetta è che siano migliori rispetto a quelle del CBOR dato che il bottleneck sembra essere la scrittura dei dati in RAM. In questo caso, infatti, i pacchetti generati sono più piccoli, pertanto si avranno minori ritardi dovuti alla scrittura.

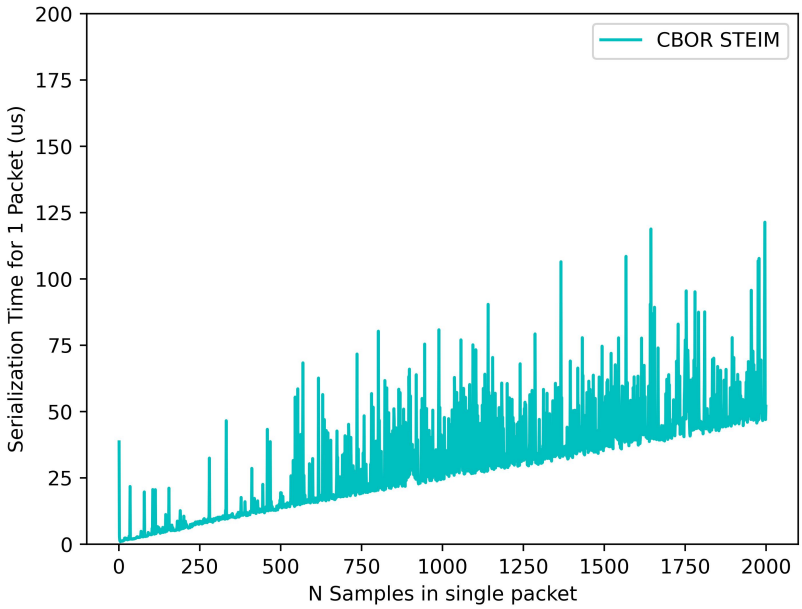

Figura 4.22: Andamento delle tempistiche di serializzazione in funzione del numero di campioni in ogni pacchetto per il formato CBOR con STEIM2.

La misura sembra essere molto "rumorosa", in Figura 4.23 viene calcolato il tempo medio per la serializzazione di un pacchetto effettuando la misura su un chunk di 5000 pacchetti. Il risultato viene inoltre messo a confronto con i risultati ottenuti dai formati classici.

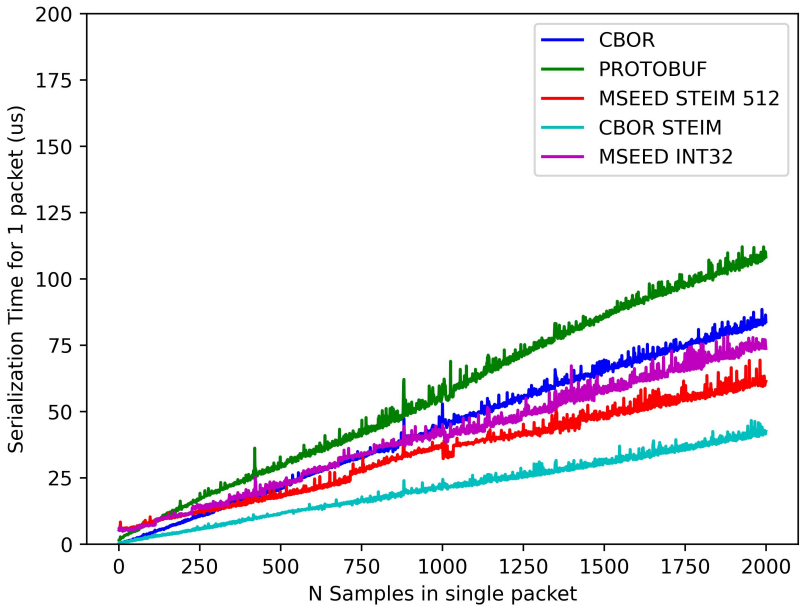

Figura 4.23: Tempistiche medie di serializzazione in funzione del numero di campioni in ogni pacchetto, tutti i formati.

Vengono inoltre riportate le linee di tendenza per ciascun andamento in Figura 4.24

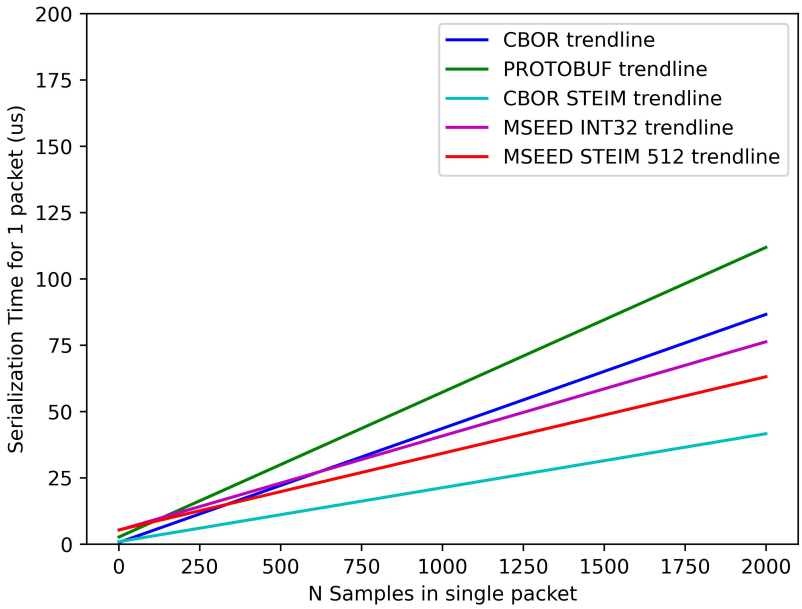

Figura 4.24: Linee di tendenza delle tempistiche medie di serializzazione in funzione del numero di campioni in ogni pacchetto, tutti i formati.

Capitolo 5

Conclusioni

A seguito della caratterizzazione fatta sulle onde sismiche e di una approfondita analisi sui sistemi di Earthquake Early Warning Systems, si giunge alla conclusione che la soluzione ottimale è quella di diffondere in maniera quanto più uniforme possibile sensori IoT connessi in rete, in grado di campionare forme d'onda generate da movimenti della crosta terrestre ed inviarle ad un sistema centrale per analizzare i dati e lanciare un eventuale allarme alla popolazione.
I sensori IoT, per loro definizione, rispecchiano tutte le principali caratteristiche necessarie ad un sistema di earthquake early warning, in quanto sono piccoli, hanno bisogno di poca energia, sono economici e possono essere gestiti in grande quantità.

Il principale problema nello sviluppo di un earthquake early warning system IoT consiste nel riadattare tecnologie pre-esistenti e protocolli ai nodi sensore, che generalmente sono composti da un microcontrollore. È per questo che sono stati studiati e sviluppati formati alternativi a quelli classici.
Sostituendo SEEDLink (formato standard utilizzato per la trasmissione di dati sismici) con MQTT si ha maggiore interoperabilità tra tutti i dispositivi, si è quindi in grado implementare il sistema su quasi qualsiasi architettura di PC o microcontrollori.

Inoltre, grazie al formato proposto, migliore degli altri sia sotto il punto di vista dei tempi di serializzazione che sotto quello della dimensione dei pacchetti generati, è possibile pensare serializzare pacchetti su nodi sensori di qualsiasi genere.
Il software necessario per generare pacchetti CBOR con codifica STEIM2 è leggero e, a differenza del software necessario per generare pacchetti MiniSEED, può essere eseguito sulla quasi totalità delle architetture e dei sistemi operativi.

Infine, tenendo conto che:

- MQTT è in grado di trasportare payloads anche molto grandi;
- Il formato di serializzazione proposto può essere usato indistintamente con pacchetti sia piccoli che molto grandi;

Si può pensare di abbinarli per sfruttare in maniera congiunta entrambe le proprietà. Si ottiene così un sistema scalabile ed estremamente elastico.

Ringraziamenti

Gentili tutti,

Con grande gioia e gratitudine, vorrei dedicare alcuni ringraziamenti speciali alle persone che hanno contribuito al completamento della mia tesi e al mio percorso accademico. Senza il loro sostegno, supporto e incoraggiamento, questo traguardo non sarebbe stato possibile.

Innanzitutto, desidero ringraziare la mia stimata professoressa e relatrice, Paola, per la sua guida preziosa, la sua competenza e la sua pazienza nel corso di questa esperienza. I suoi consigli e le sue correzioni mi hanno aiutato a migliorare e a dare il meglio di me stesso.

Un ringraziamento speciale va anche a Marco e Alberto, che mi hanno seguito attentamente nello sviluppo della mia tesi. La loro competenza e il loro supporto costante mi hanno fornito una solida base su cui costruire il mio lavoro.

Vorrei esprimere la mia profonda gratitudine alla mia famiglia, in particolare ai miei genitori, per il loro sostegno incondizionato e per aver creduto in me in ogni fase del mio percorso. Il vostro amore, il vostro incoraggiamento e i vostri sacrifici mi hanno dato la forza di superare ogni ostacolo.

Un ringraziamento speciale va alla mia fidanzata Giorgia, che è stata un'ancora di stabilità e sostegno durante i momenti difficili. Grazie per aver soppor-

tato le mie lamentele, per avermi incoraggiato e per essere stata al mio fianco in ogni fase del mio percorso.

Non posso dimenticare di menzionare il mio caro amico Riccardo, conosciuto affettuosamente come Ritardo. La tua amicizia è stata un dono prezioso, e sei stato un compagno di avventure e di crescita come uomo di scienza. Grazie per i tuoi consigli e per le tue conversazioni stimolanti.

Desidero anche estendere la mia gratitudine ai miei zii, cugini e nonni, in particolare a nonno Tony. Grazie per avermi introdotto al meraviglioso mondo dell'ingegneria facendomi smontare cose fin da bambino. Il vostro appoggio e la vostra presenza hanno avuto un impatto significativo sulla mia passione per la scienza e per l'apprendimento.

Infine, vorrei ringraziare tutti i miei amici, che hanno reso il mio percorso universitario più ricco e significativo. Le serate trascorse insieme, anche durante i periodi più impegnativi, mi hanno dato la forza di affrontare le sfide e mi hanno ricordato l'importanza dell'equilibrio tra studio e vita sociale.

Ringrazio ancora una volta ognuno di voi per il vostro contributo, grande o piccolo che sia. Senza il vostro aiuto, il mio traguardo non sarebbe stato raggiunto. Siete stati una parte fondamentale di questa avventura e vi porterò sempre nel mio cuore.

Con gratitudine sincera,
Tomas

Bibliografia

[1] Carsten Bormann. *RFC 8949 Concise Binary Object Representation (CBOR)*. 2020. URL: https://www-rfc--editor-org.translate.goog/rfc/rfc8949?_x_tr_sl=it&_x_tr_tl=en&_x_tr_hl=it&_x_tr_pto=wapp (visitato il 06/10/2023).

[2] IRIS EDU. *SEED DATA FORMAT*. 2000. URL: http://ds.iris.edu/ds/nodes/dmc/data/formats/seed/ (visitato il 06/10/2023).

[3] FSDN. *SEED Reference manual*. 2012. URL: http://www.fdsn.org/pdf/SEEDManual_V2.4.pdf (visitato il 06/10/2023).

[4] Colombelli S. Caruso A. Zollo A. Festa G. e Kanamori H. "A P wave-based, on-site method for earthquakeearly warning. (Italia) [On eophysical Research Letters]". In: *ournal of Geophysical Research* (2014). DOI: http://dx.doi.org/10.1002/andp.19053221004.

[5] INGV. *Earthquake Early Warning*. 2020. URL: https://www.ct.ingv.it/osuct/index.php/it/news/37-earthquake-early-warning-eew (visitato il 06/09/2023).

[6] Inc. Joseph M. Steim Quanterra. "'Steim' Compression". In: *nd* (1994). DOI: nd.

[7] Caruso A. Colombelli S. Elia L. Picozzi M. e Zollo A. "An on-site alert level early warning system for Italy,. (Italia) [On ournal of Geophysical Research]". In: *ournal of Geophysical Research* (2017). DOI: http://dx.doi.org/10.1002/andp.19053221004.

[8] MQTT. *MQTT: the standard for IoT messaging*. 2022. URL: https://mqtt.org (visitato il 06/10/2023).

[9] PRESTo. *What is PRESTo?* 2020. URL: http://www.prestoews.org/about.php (visitato il 06/09/2023).

[10] Adam T. Ringler e John R. Evans. "A quick seed tutorial. (Italia) [On]". In: *A quick seed tutorial* (2015). DOI: 86:1717âĂŞ1725.

[11] Srđan Popić; Dražen Pezer; Bojan Mrazovac; Nikola Teslić. "Performance evaluation of using Protocol Buffers in the Internet of Things communication. (Italian) [On]". In: *Performance evaluation of using Protocol Buffers in the Internet of Things communication* (2016). DOI: 10.1109/SST.2016.7765670.

[12] Somayya Madakam; R. Ramaswamy; Siddharth Tripathi. "Internet of Things (IoT): A Literature Review. (English) [On]". In: *Journal of Computer and Communications,03,164-173* (2015). DOI: 10.4236/jcc.2015.35021.

[13] A. Zacchiilli. *tudio e sviluppo di protocolli real-time per sistemi di Early Warning*. 2021. URL: https://tesi.univpm.it/handle/20.500.12075/7361 (visitato il 06/10/2023).