ALTRI PARADIGMI DI PROGRAMMAZIONE

A.A. 2020/2021

Laurea triennale in Informatica

19: Esempi svolti 4

RETI COMPLESSE

Per concludere la panoramica sui metodi "tradizionali" di costruzione di programmi distribuiti, vediamo un ulteriore esempio che coinvolge nodi con caratteristiche diverse che interagiscono fra loro con un paradigma diverso dalla semplice richiesta/risposta.

CONTACT-TRACING

Uno degli strumenti fondamentali per contrastare la diffusione della pandemia da COVID-19 è l'individuazione precoce dei contatti avuti dalle persone contagiose, in modo da prevenire ulteriori contagi.

Sfruttando l'ubiquità dei dispositivi mobili con capacità bluetooth, si è pensato di sfruttare tale tecnologia per questo scopo.

"Decentralized Privacy-Preserving Proximity Tracing" propone un protocollo, basato sulla tecnologia Bluetooth, in grado di soddisfare i requisiti di tracciamento preservando nel contempo le esigenze di privacy.

Il metodo si basa sull'emissione di messaggi crittografati in modo da rimanere anonimi, ma che possono essere identificati quando una parte pubblica della chiave viene rivelata.

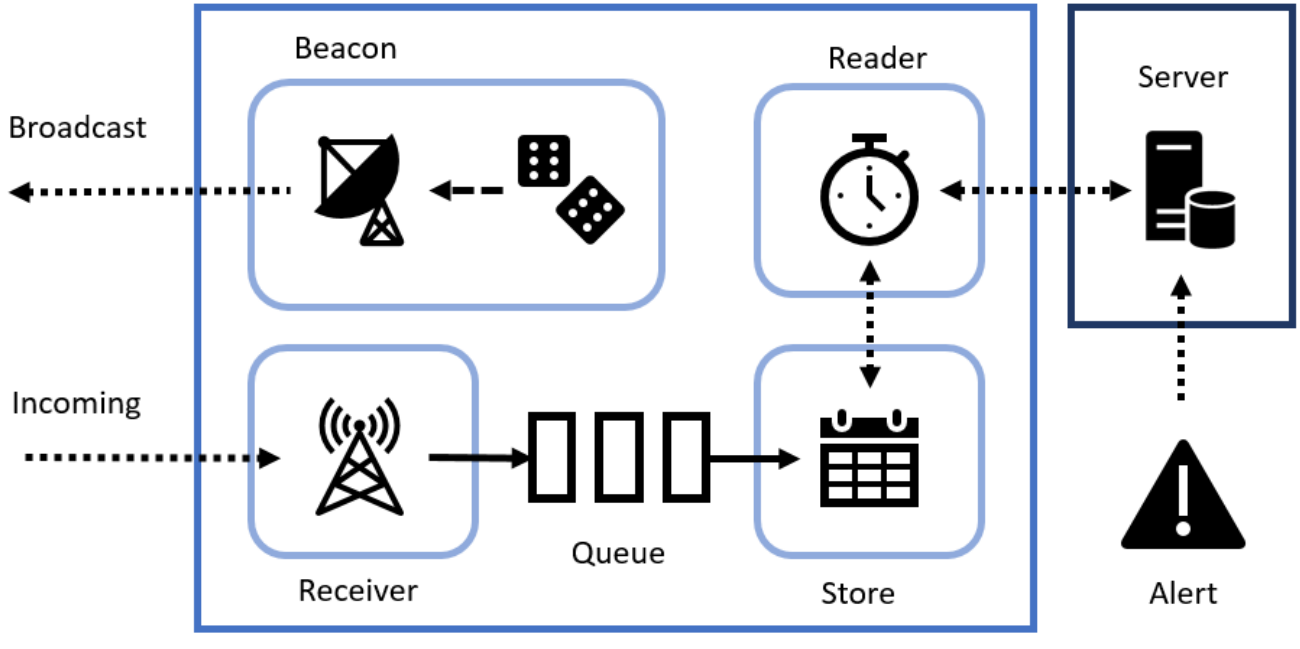
In questo modo ciascun dispositivo può controllare se una chiave pubblicata ha emesso qualcuno dei messaggi che ha ricevuto, ma non può sapere a chi appartiene né ottenere altre informazioni.

La privacy è inoltre garantita dal fatto che non esiste un deposito centrale dei messaggi, ma solo un'autorità che pubblica le chiavi appartenenti a chi è risultato positivo, fornite personalmente da quest'ultimo.

Questo funzionamento decentralizzato e anonimizzato fa si che anche gli attacchi più teorici richiedono risorse considerevoli per essere messi in pratica.

E' interessante vedere come si può implementare un sistema vagamente simile nel funzionamento per osservare in pratica i comportamenti di un insieme di agenti indipendenti e per sfruttare alcune caratteristiche dei protocolli di rete.

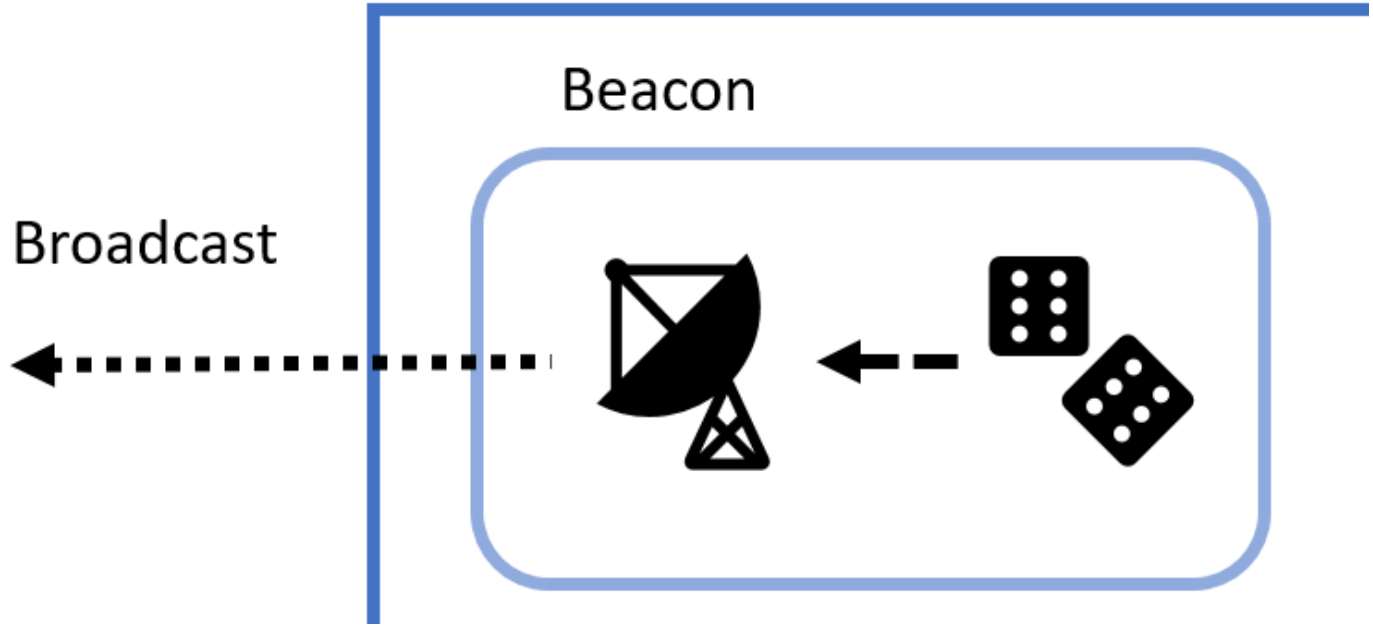
Un sistema come il dp-3T è strutturato vagamente in questo modo:



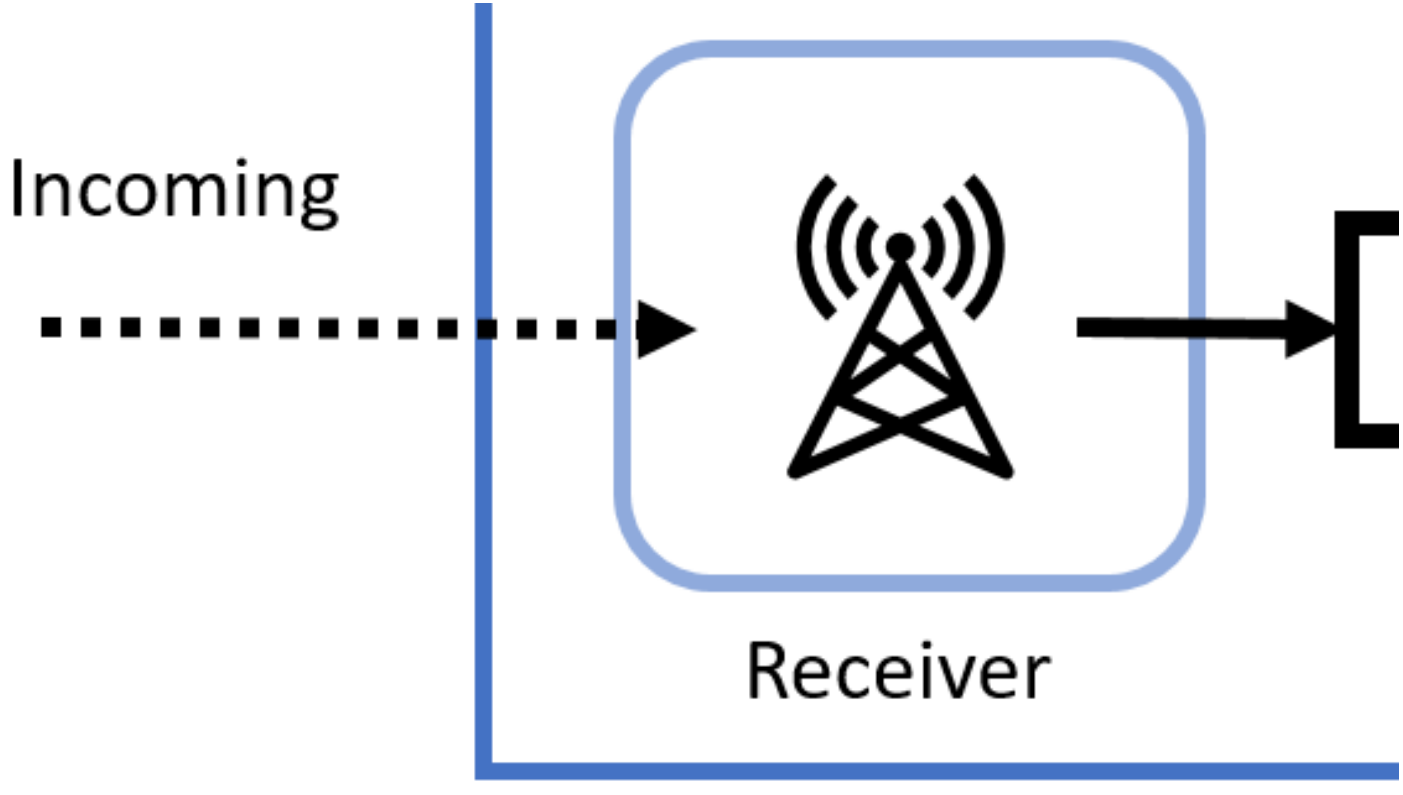
Il sistema è composto di uno o più nodi, che rappresentano le singole persone interessate dal tracciamento. Ogni nodo è diviso in componenti funzionali indipendenti e concorrenti.

Un server centralizzato raccoglie e distribuisce le segnalazioni.

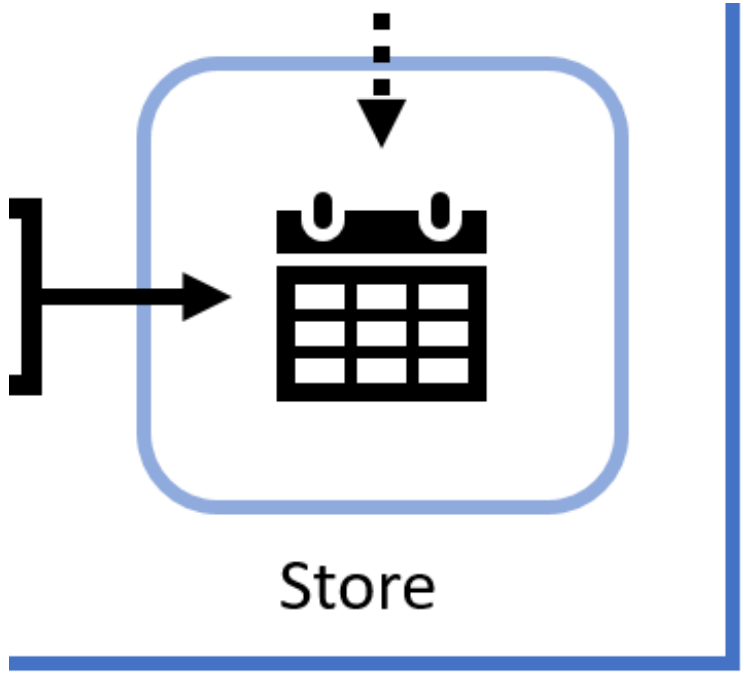
Il componente *Beacon* emette dei messaggi broadcast indirizzati a tutti gli indirizzi della rete cui appartiene.



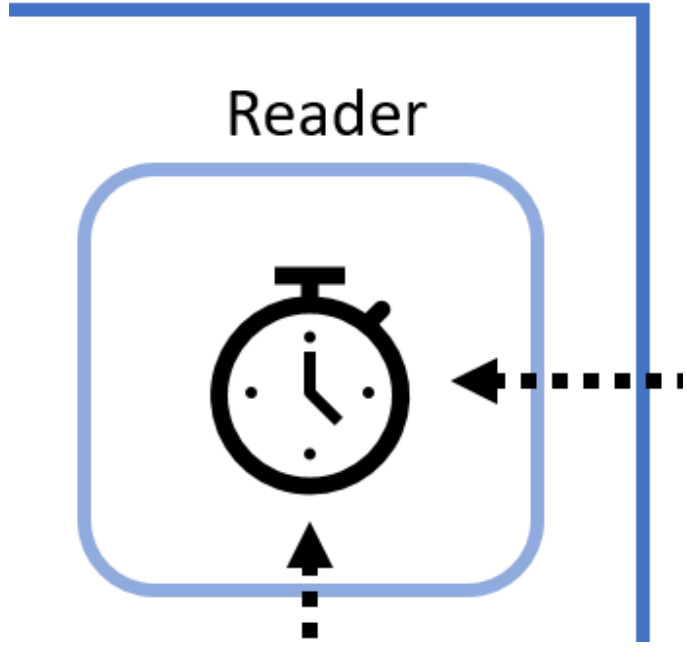
Il componente *Receiver* ascolta sulla porta indicata, e accoda ogni messaggio ricevuto.



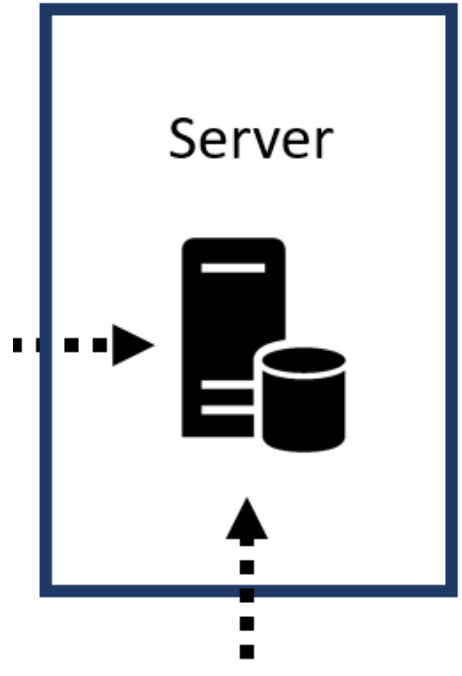
Il componente *Store* conserva i messaggi ricevuti, e permette di controllare se un messaggio segnalato è stato notato in passato.



L'ultimo componente del nodo, il componente *Reader*, a intervalli regolari interroga il servizio centrale e legge i messaggi segnalati.



Abbiamo bisogno anche di un server centrale, in grado di pubblicare una lista di messaggi segnalati.



IMPLEMENTAZIONE

BEACON

Il componente Beacon deve, ad intervalli regolari, emettere messaggi a tutti i nodi in ascolto sulla stessa rete.

Fra le primitive che abbiamo a disposizione, l'unica con queste caratteristiche è il Datagram.

Il Beacon dovrà essere un Thread indipendente, con un ciclo che prepara, emette il messaggio e poi aspetta un tempo stabilito.

RECEIVER

Il componente Receiver deve creare e rimanere in ascolto del DatagramSocket. Quando riceve un messaggio, lo pone sulla coda per la sua elaborazione.

Anche in questo caso servirà un Thread concorrente, che in un loop controllato da un flag di chiusura, attende l'arrivo di un messaggio e ne estrae il contenuto.

STORE

Il componente Store deve mantenere un insieme di messaggi già visti in modo che possano essere consultati per identificare un contatto.

Per non interferire con la ricezione dei messaggi, questo componente li preleva dalla coda e li aggiunge all'insieme del messaggi ricevuti.

La struttura dati utilizzata è thread-safe in modo che il test di appartenenza per il controllo se un messaggio è stato ricevuto può essere fatto da un altro thread.

READER

Il componente Reader deve collegarsi al sistema centrale, scaricare la lista dei messaggi segnalati e controllare se uno o più di essi sono stai ricevuti.

Un timer controlla il recupero periodico della lista da una url configurata all'avvio. Ottenuto l'elenco, ogni messaggio viene cercato nella struttura dati dello Store per verificare l'eventuale ricezione.

La lista è pubblicata come testo semplice, un messaggio per riga. E' quindi semplice da leggere ed interpretare.

SERVER

Il server centrale deve fornire la lista dei messaggi segnalati, e permettere di aggiungere nuovi messaggi a tale lista.

Per semplicità, è realizzato come un servizio web che ad una richiesta GET fornisce i messaggi segnalati come righe di un testo semplice, e con un POST accetta nuovi messaggi da considerare segnalati.

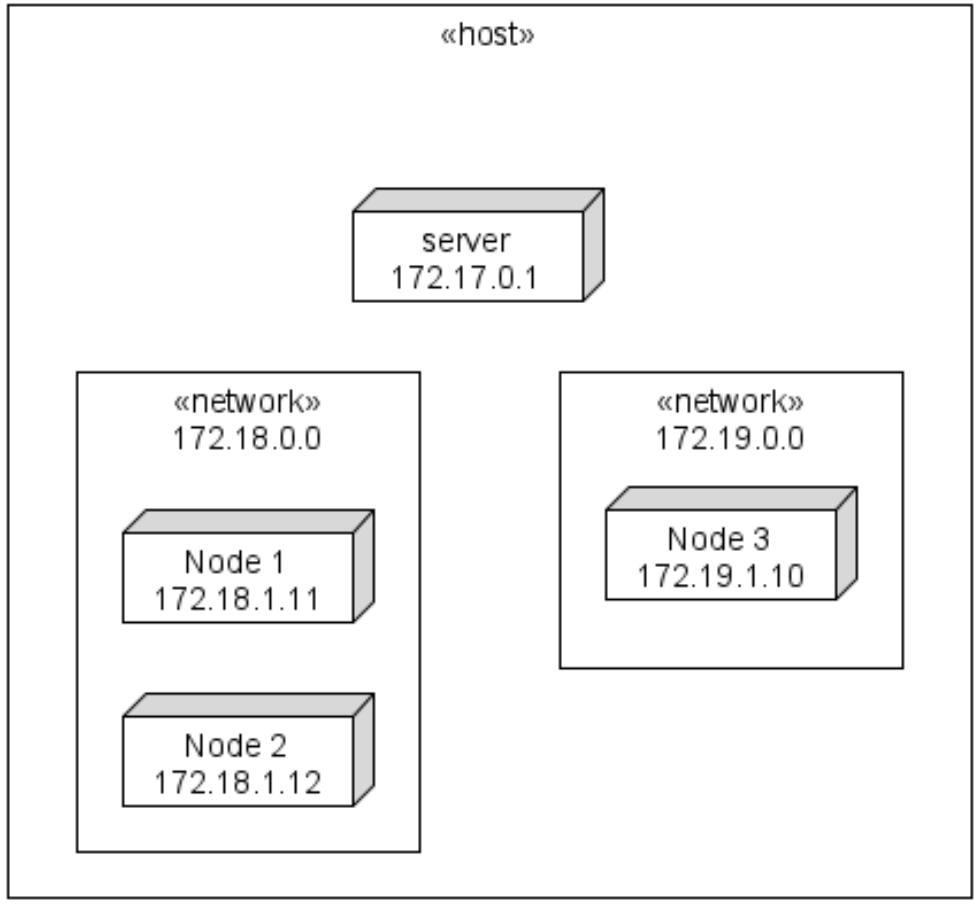
E' implementato come un semplice server VertX, con una struttura dati thread-safe per permettere le operazioni di lettura e scrittura in modo asicrono.

NETWORKING

Per verificare il funzionamento della trasmissione dei messaggi in broadcast, abbiamo bisogno di strutturare la rete in modo da mettere alcuni nodi in una rete dove il messaggio broadcast può circolare, ed altri in una rete separata.

Per raggiungere questo risultato con risorse limitate, il progetto del singolo nodo è in grado di produrre una immagine Docker.

Usando le opzioni di networking di questo strumento, possiamo realizzare i nostri scopi.


«host»
server
172.17.0.1
«network»
172.18.0.0
Node 1
172.18.1.11
Node 2
172.18.1.12
«network»
172.19.0.0
Node 3
172.19.1.10

I nodi 1 e 2 si trovano sulla stessa sottorete: ciascuno di essi riceverà i messaggi broadcast dell'altro.

Il nodo 3 è in un'altra sottorete, e non riceverà i messaggi degli altri 2.

Tutti e tre raggiungono il server allo stesso indirizzo, sull'host.

Repository ct-node

https://hg.sr.ht/~michelemauro/app2020-ctnode

Repository ct-server

https://hg.sr.ht/~michelemauro/app2020-ctserver

LINK UTILI

COVID-19: le App di tracciamento nella fase due in Italia e in Europa - Avv. Manuela Soccol

https://bit.ly/2VwdWVz