Appunti Progetto ANAWS

Caratteristiche Attuali dell’estensione Observe

* Prevede l’affidabilità end-to-end usando il messaggio CON di CoAP
* Per la tempestività invece prevede l’uso di un’opzione che indica la validità di una notifica durante un periodo di tempo, usando il modello di caching offerto da CoAP (da riguardare), anche se non garantisce la possibilità di decidere a priori per quanto tempo un singolo messaggio può essere considerato valido.
* Da facoltà al server (il publisher) di decidere in quale ordine le notifiche devono arrivare ai client, ovvero dare priorità alle notifiche, per venire incontro al fatto che i nodi WSN possono avere ruoli e requisiti differenti.

Estensione sviluppata

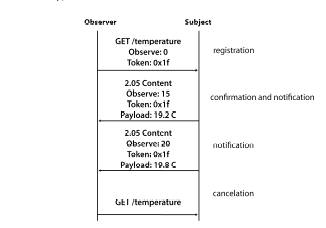
* Si basa sulla priorità di invio
* I client possono esprimere la priorità con cui vogliono ricevere le notifiche
* Il server può accettare la richiesta del client oppure negoziarla
* Le priorità si dividono in due categorie
* Critiche
* Non critiche

Modello

* Formato da due moduli, l’osservatore ed il soggetto
* L’osservatore è un client interessato a ricevere notifiche quando cambia lo stato di una risorsa
* Il soggetto invece è il server che gestisce queste notifiche
* In questo modello non è prevista la presenza del broker
* È comunque prevista la presenza di un intermediario per migliorare la stabilità, che riceve le richieste di registrazione dell’osservatore e invia una richiesta di registrazione al soggetto.
* Questo processo deve essere trasparente e supportare reti su larga scala e formate da percorsi multi-hop, (dove quindi in percorso di n nodi compreso osservatore e soggetto ci saranno n-2 nodi intermedi)

Processo di registrazione e notifica

* Per inviare una registrazione l’osservatore invia una richiesta GET indicando nel campo *observe* il valore 0 ed un token per identificare la richiesta
* Nelle notifiche invece il campo *observe* sarà diverso da 0 ed indicherà il numero di sequenza della notifica, mantenendo invece sempre il campo token invariato
* Il soggetto nel momento della registrazione può decidere se rifiutare la richiesta dell’osservatore inviando la risposta priva del campo *observe.*
* L’osservatore può in qualunque momento decidere di smettere di ricevere notifiche semplicemente rispondendo ad una notifica con un messaggio RST oppure inviando un messaggio GET per la stessa risorsa.



* Per garantire l’affidabilità si fa uso dei meccanismi definiti in CoAP: un nodo che riceve un messaggio CON deve notificare che ha ricevuto correttamente il messaggio e talora il mittente non riceva un ACK, esso ritrasmetterà la richiesta. Il processo di ritrasmissione segue il modello STOP-AND-WAIT con un backoff esponenziale.
* Un nodo può anche scegliere di inviare messaggi in modo non affidabilie usando dei messaggi NON. In questa estensione la scelta del tipo di messaggio (CON o NON) è a carico del soggetto.
* Per la tempestività dei messaggi invece è previsto l’uso della validità temporale, avvalendosi del meccanismo di cache presente in CoAP
  + In CoAP è presente un parametro MaxAge che viene usato per indicare la validità temporale
  + Un osservatore può salvare in cache un messaggio e riutilizzarlo per future richieste finché la sua validità temporale sarà rispettata

# Differenziazione QoS

* L’osservatore può richiedere un certo livello di priorità per l’invio delle notifiche
  + Bassa, media, alta e massima priorità
    - I primi ad essere serviti sono i nodi con la priorità massima
    - Il livello alto è richiesto dai nodi che vogliono ricevere notifiche quando un evento critico è rilevato o quando finisce, le notifiche vengono spedite immediatamente dagli osservatori con la massima priorità
    - Gli osservatori che sono interessati sia alle notifiche critiche che alle non critiche possono scegliere un livello di priorità medio o basso, con il primo che ha la precedenza sul secondo. Se ci sono più osservatori con la stessa priorità, l’ordine della spedizione delle notifiche sarà uguale all’ordine con cui sono state fatte le richieste
  + Andando più nel dettaglio i livelli sono caratterizzati nel seguente modo:
    - Livello più basso (1): Il soggetto invia sia le notifiche non critiche che quelle critiche con bassa priorità, con il valore QoS impostato a 00
    - Livello medio (2): il soggetto invia sia le notifiche non critiche che quelle critiche con media priorità, impostando il valore QoS a 01
    - Livello alto (3): Il soggetto invia solo le notifiche critiche con la priorità alta, impostando il campo QoS a 10
    - Livello massimo (4): Il soggetto invia le notifiche con la massima priorità solo all’inizio e alla fine di un evento critico, impostando il campo QoS a 11
* Il modello prevede che l’osservatore ed il soggetto possano negoziare il livello di qualità del servizio, con il soggetto che ha la possibilità di negare o negoziare la richiesta fatta dall’osservatore, nel caso in cui non abbia sufficienti risorse per gestire tutte le richieste.
* Se il soggetto accetta la richiesta dell’osservatore, allora gli risponde impostando il campo *observe* con la richiesta dell’osservatore, altrimenti può proporre un nuovo valore all’osservatore, che a sua volta potrà scegliere se accettarlo, rispondendo con il valore scelto dal soggetto, oppure rifiutarlo scartando il messaggio.

# File Importanti

[RFC di riferimento](https://tools.ietf.org/html/rfc7641#page-9)

* Er-coap-observe.c ( subject ):
  + mantiene la lista degli osservatori
  + notifica lo stato della risorsa passata come parametro a tutti gli osservatori interessati
  + Gestisce la registrazione degli osservatori ( coap\_observe\_handler ? )
* Er-coap.c: implementa le funzioni di base del protocollo COAP, tra cui:
  + Coap\_init\_message( void\* packet, coap\_message\_type\_t type, uint8\_t code, uint16\_t mid ) che inizializza un messaggio coap, type può essere NON o CON e code è un codice di request o response contenuti nel file er-coap-constants.h
  + Coap\_set\_header\_observe(void\* packet, uint32\_t observe) abilita l’opzione COAP\_OPTION\_OBSERVE e nel campo dell’opzione un intero che nelle request è sempre 0, mentre nelle response è diverso da 0 e serve da sequence number. Si può sfruttare questo come campo QoS field? Si perde la funzione di sequence number nelle response ?

|  |  |  |
| --- | --- | --- |
| * + - Livello Priorità | * + - Qos Field ( HEX ) | * + - Bit rimanenti ( HEX ) |
| * + - 00 | * + - 00 | * + - 000000 |
| * + - 01 | * + - 40 | * + - 000000 |
| * + - 10 | * + - 80 | * + - 000000 |
| * + - 11 | * + - C0 | * + - 000000 |

* + Coap\_get\_header\_observe( void\* packet, uint32\_t observe ) controlla che l’opzione sia abilitata e mette nel parametro observe il campo packet->observe.
  + Coap\_set\_payload( void\* packet, const void\* payload, size\_t length )
* Er-coap-constants:
  + contiene i codici di risposta e richiesta
  + Il pacchetto nell’header ha una bitmap che consente di capire quali opzioni sono abilitati ( l’opzione observe è il sesto bit )
* Er-coap-observe-client ( observer ):
  + estende l’implementazione Erbium per abilitare i client osservatori
  + Ha molte funzioni identiche a quelle di er-coap-observe.c che servono a mantenere una lista di soggetti
  + Gestisce la richiesta di registrazione e la risposta
  + Gestisce le notifiche delle risorse osservate

Per testare:

* Er-example-server.c : offre un servizio RESTful esponendo le risorse disponibili usando l’implementazione Erbium di contiki
* Er-example-client.c : inizializza un coap\_engine che sta a riceve tutti i messaggi coap, il resto credo sia la registrazione sul soggetto che possiede la risorsa che vuole osservare