TESI DI LAUREA

# Studio di interoperabilità tra i sistemi operativi Contiki e TinyOS per reti di sensori wireless

RELATORE: Ch.mo Prof. Michele Zorzi

CORRELATORI: Ing. Paolo Casari, Ing. Angelo P. Castellani

LAUREANDO: Renato Spiandorello

Corso di laurea Vecchio Ordinamento in Ingegneria Elettronica

Anno Accademico 2008-2009

Padova, 27 Aprile 2010

# Sommario

Le reti di sensori wireless in futuro potranno candidarsi per l'estensione della rete Internet agli oggetti, dotandoli di capacità di comunicazione IP.
In ambito accademico ed industriale, si stanno diffondendo alcuni sistemi operativi che condividono lo stesso hardware e gli stessi protocolli di rete.
E' inevitabile che la diffusione delle reti di sensori wireless richieda la verifica dell'interoperabilità tra le varie realizzazioni, partendo dal networking fino ad arrivare al livello applicativo.
Questo lavoro si focalizza sullo studio del sistema operativo Contiki, sviluppato da SICS, Swedish Institute of Computer Science e sulla verifica di interoperabilità a livello di networking con il sistema operativo TinyOS, sviluppato dal gruppo WEBS di Berkeley e molto diffuso in ambiente accademico.
In questo lavoro si verificherà in particolare l'interoperabilità delle implementazioni del protocollo 6lowpan che consente l'utilizzo di IPv6 su reti LR-WPAN.

# Indice

# 1. Wireless sensor networks

## 1.1 Introduzione

Una rete di sensori è un insieme di piccoli dispositivi posizionati in prossimità oppure all'interno del fenomeno da osservare. I sensori possono rilevare grandezze come temperature, umidità, pressione, luce ed anche movimenti di oggetti, livello di rumore ed altre ancora.
I progressi tecnologici hanno consentito lo sviluppo di sensori costituiti da apparecchi a bassa potenza, dal costo di produzione ridotto e capaci di rilevare grandezze fisiche e di comunicare tra loro tramite tecnologia wireless a raggio limitato.
I nodi sono provvisti di un processore on-board che permette di effettuare semplici elaborazioni sui dati grezzi ed inviare solo i dati interessanti a nodi di raccolta, detti nodi sink. Le reti di sensori wireless possono essere utilizzate in molte applicazioni, che vanno da quelle in ambito militare, scientifico, industriale, medico, domestico e dialogano tra loro mediante alcune tecniche utilizzate nelle reti wireless ad hoc.
Le reti di sensori possono garantire una buona qualità di informazioni con elevata fedeltà in tempo reale, da ambienti ostili, riducendo i costi di trasmissione delle informazioni stesse, garantendo l'integrità per un periodo di tempo più lungo possibile, anche in caso di attacco alla rete da parte di organi esterni o problemi di tipo hardware.
L'utente che necessità di informazioni dal wireless sensor network interpella il nodo sink, il quale effettua una interrogazione nella rete e riceve una o più risposte, che vengono elaborate prima di restituire la risposta all'utente.
Le reti di sensori sono modellate come un database distribuito che si basa su nodi attivi con informazioni ottenibili tramite un linguaggio simile ad SQL, infatti il nodo sink non può collezionare informazioni grezze direttamente da ciascun nodo, a causa del numero elevato di nodi e per la distanza tra i nodi ed il nodo sink.
La posizione dei nodi all'interno della rete non è predeterminata, perciò gli algoritmi ed i protocolli utilizzati devono possedere caratteristiche auto-organizzative, ottimizzati per ridurre il consumo di energia.
La realizzazione di una rete di sensori wireless richiede l'utilizzo di tecniche di rete specifiche che derivano dalle reti ad-hoc, cioè dalle reti che non prevedono un coordinatore centrale.
In particolare l'elevata densità dei nodi permette di utilizzare algoritmi di rete multi-hop, con trasmissioni a bassa potenza.
Lo sviluppo di sensori wireless ha visto l'affermazione delle LR-WPAN, Low rate Wireless Personal Area Network (di seguito LoWPAN), caratterizzate da dimensioni contenute ed al contempo bassi transfer rate.
Le reti LoWPAN si inseriscono all'interno di una generica classificazione delle reti in base all'area di copertura, in termini di dimensione fisica, non di densità di sensori.

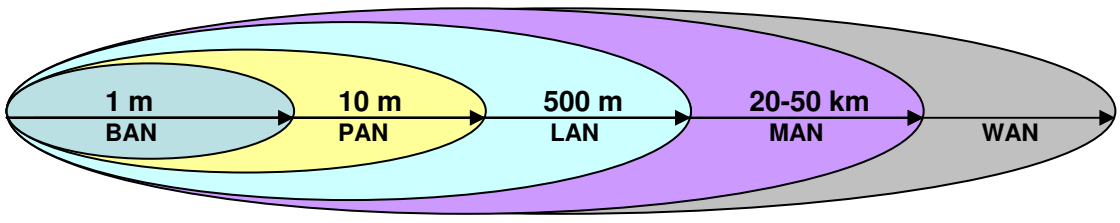


Figura 1.1 Classificazione per area di copertura

Per quanto riguarda le PAN, cioè le reti personali che devono coprire distanza fino a 10 m, il comitato dedicato alla WPAN (Wireless Personal Area Network), indicato come IEEE 802.15, ha definito tre distinte classi di WPAN caratterizzate da diversi data rate, consumi e QoS:

- WPAN ad elevati data rate, IEEE802.15.3, all'interno del quale si trovano tecnologie dedicate ad applicazioni multimediale che richiedono elevato QoS, tra le altre WiMedia e Bluetooth2.
- WPAN a medio data rate, IEEE 802.15.1, pensate per la sostituzione dei cavi per l'elettronica di consumo, dove troviamo Bluetooth.
- WPAN a bassi data rate, IEEE 802.15.4, che servono a rispondere alle esigenze di bassi consumi e bassi costi.

In questo elaborato verranno utilizzati sensori sperimentali Telos B, progettati da UC Berkley e disponibili commercialmente tramite Crossbow, che utilizzano lo standard IEEE 802.15.4 per le comunicazioni radio.
Una rete di sensori necessita di uno spazio di indirizzamento ampio e l'autoconfigurazione dei sensori risulta fondamentale: queste richieste rendono IPv6 adeguato per la realizzazione del layer 3.

## 1.2 IEEE 802.15.4

Lo standard IEEE 802.15.4 definisce le tecniche di trasmissione radio a bassa potenza in ambito wireless personal area networks (WPANs).
IEEE 802.15.4 definisce sia il layer PHY che MAC ed è largamente utilizzato da molti specifiche come ad esempio 6LoWPAN, ISA100 e ZigBee.
IEEE 802.15.4 a livello PHY utilizza DSSS e OQPSK per l'invio di simboli a 4 bit, ad una speed massima di 62,5 ksimboli o 250 kbps, è disponibile in tre frequenze, suddivise in canali, una delle quali è globale e le altre sono locali, il bit rate varia da 20 a 250 kbit/s.

| **Frequenza (Mhz)** | **Region** | **Channel numbers** | **Bit rate (kbit/s)** |
|---|---|---|---|
| 868 | Europe | 0 | 20 |
| 902-928 | US | 1-10 | 40 |
| 2400-2483,5 | Worldwide | 11-26 | 250 |

A livello MAC, IEEE 802.15.4 divide i nodi sullo stesso canale in multiple Personal Area Networks (PANs) e non permette a nodi che appartengono a PAN diversi di dialogare sullo stesso canale.
L'indirizzamento può essere a 64 bit o 16 bit, unicast e broadcast, la dimensione del pacchetto può raggiungere i 127 byte e la sicurezza del link-layer è affidata all'encryption AES a 128-bit.
La comunicazione tra i nodi può essere ottimizzata nel caso sia presente un PAN-Coordinator che può fornire un indirizzamento temporaneo a 16-bit, al posto dell'indirizzamento standard a 64-bit.
Il MAC può utilizzare il metodo beaconless o beacon-enabled: il primo metodo utilizza l'accesso al canale CSMA ed opera in modo simile ad IEEE 802.11 senza channel reservation, mentre il secondo è più complesso, utilizza un time division

multiple access (TDMA) ibrido, che prevede una struttura con superframe con la possibilità di riservare time-slot per dati critici.
IEEE 802.15.4 utilizza le seguenti frame:

- Data frame, per il trasporto dei dati.
- Acknowledment frame, che vengono inviate dal ricevente dopo 12 symbol period (192 μs), dopo la ricezione di una frame, se richieste tramite acknowledgment bit nel data frame MAC header.
- MAC layer command frames, utilizzate in modalità beacon-enabled per l'utilizzo dei servizi di livello MAC, come ad esempio l'associazione e dissociazione dal coordinatore e le gestione delle trasmissioni sincronizzate.
- Beacon frames, utilizzata dal coordinatore in beacon-enabled per la comunicazione strutturata con i nodi associati.

Di seguito si ripota la struttura del MAC layer data packet, preceduto da un preambolo e da un delimitatore iniziale non riportati nello schema:

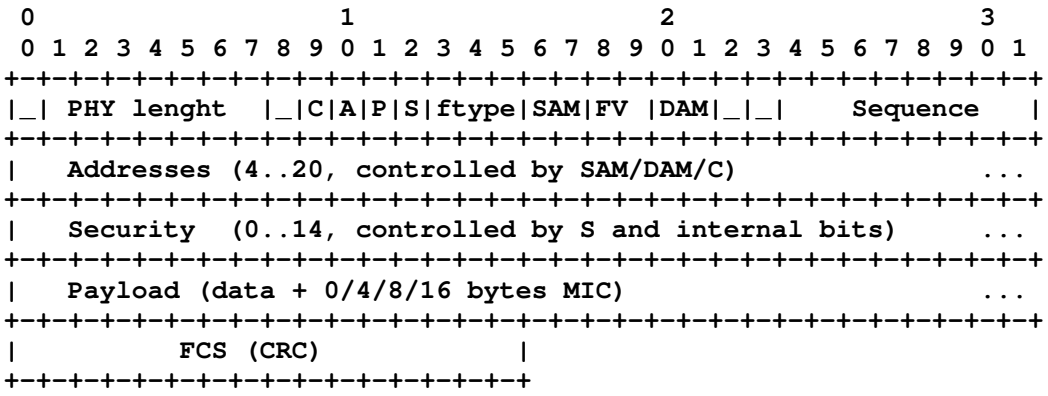


Figura 1.2 struttura MAC data packet

Il delimitatore iniziale, non riportato nello schema, viene evidenziato nelle catture che seguiranno con il valore 02.
Il data packet inizia con un bit riservato e con 7 bit di lunghezza che definiscono la dimensione della parte rimanente del pacchetto, compreso il checksum e perciò il pacchetto complessivo può essere lungo da 1 a 128 byte.
Si trovano poi 3 byte per la parte fissa del'header di livello MAC, che definiscono anche la parte variabile che segue.
In particolare, la parte variabile definita "Addresses" dipende dai valori dei campi SAM, DAM e C. SAM e DAM possono indicare l'indirizzamento di lunghezza 0 (PAN coordinator), 16-bit e 64-bit.
Nel caso di 16-bit o 64-bit, l'indirizzo è preceduto dal PAN identifier di 16 bit, tranne nel caso il bit C sia settato.
Di seguito la descrizione dei campi della parte fissa dell'header e trailer dal data packet IEEE 802.15.4.

| | |
|---|---|
| _ | riservato |
| C | compressione del PAN ID |
| A | richiesta di ACK |
| P | Frame pending |

| | |
|---|---|
| S | security attivata |
| ftype | Frame type (001 binary per data packet, 010 binary per acknowledgment) |
| SAM | Source addressing mode |
| FV | Versione della frame (00 binary per compatibilità con versione 2003 e 01 per compatibilità con versione 2006) |
| DAM | Destination addressing mode |
| Sequence | Sequence number per ACK |
| FCS | Frame check sequence |

Di seguito si ripotano le combinazioni possibili per i campi SAM e DAM

| | |
|---|---|
| 00 | Nè PAN identifier nè indirizzamento |
| 01 | riservato |
| 10 | il campo indirizzamento contiene la versione ridotta a 16-bit |
| 11 | il campo indirizzamento contiene la versione estesa a 64-bit |

Il bit S indica la presenza del security subheader, che viene riportato di seguito.

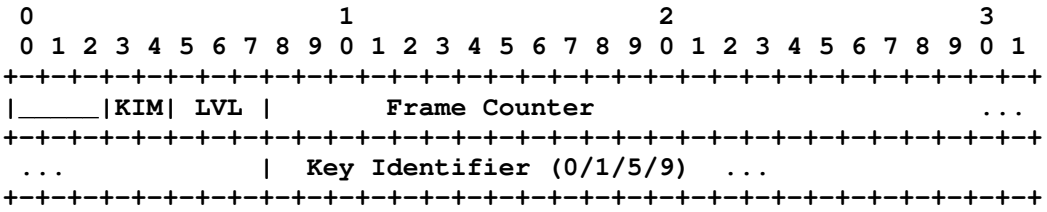


Figura 1.3 Security subheader

Nel security subheader, il campo KIM specifica la struttura del campo key identifier.

| | |
|---|---|
| 00 | key identificata da source e destination |
| 01 | key identificata da macDefaultkeysource + one-byte key index |
| 10 | Key identificata da 4 byte key source + one-byte key index |
| 11 | Key identificata da 8 byte key source + one-byte key index |

Il campo security level LVL specifica le funzioni di security impiegate ed i paramentri per l'eventuale message integrity check (MIC).

| | |
|---|---|
| 000 | Nessuna security |
| 001 | 4 byte MIC |
| 010 | 8 byte MIC |
| 011 | 16 byte MIC |
| 100 | Solo encryption |
| 101 | Encryption + 4 byte MIC |
| 110 | Encryption + 8 byte MIC |
| 111 | Encryption + 16 byte MIC |

## 1.3 Analisi di una frame IEEE 802.15.4

Di seguito si ripota l'analisi di una tipica frame IEEE 802.15.4 catturata tramite l'applicazione IPBasestation di TinyOS e che si incontrerà spesso in questo documento:

frame IEEE 802.15.4:
02 24 41 88 64 22 00 05 00 17 67 03 F4 67 17 00 05 60 10 34 8D 53 65 6E 64 65 72 20 73 61 79 73 20 48 69 21 0F EB

primo byte: 0x02
lunghezza: 0x24, 36 byte

header: 0x41 88 64 22 00 05 00 17 67
_CAPSftype: 0x41
C=1, compressione del PAN ID
A=0, non richiede ACK
P=0, no frame pending
S=0, no security
ftype=001, data packet

SAMFVDAM_: 0x88
SAM=10, 802.15.4 source address a 16 bit
FV=00, frame compatibile 2003
DAM=10, 802.15.4 destination address a 16 bit

sequenza della frame: 0x64

PAN ID: 22 00 (00 22)
802.15.4 destination address: 05 00 (00 05)
802.15.4 source address: 17 67 (67 17)

payload: 0x03 F4 67 17 00 05 60 10 34 8D 53 65 6E 64 65 72 20 73 61 79 73 20 48 69 21

CRC: 0x0F EB

## 1.4 Telos rev. B

TelosB mote (TPR240) è una piattaforma open source, costruita per scopi di sperimentazione della comunità di ricerca, progettato da UC Berkley e prodotto commercialmente da Crossbow [9] oppure con il nome Tmote Sky da Mote*iv* [10].

Figura 1.4 Telos rev. B

Il TPR2400 (codice del Telos rev. B) raggruppa tutto quello che serve per l'ambiente di laboratorio, di seguito le principali caratteristiche:

- MCU TI MSP430 8 Mhz (16-bit RISC) con 10 KB RAM
- programmabilità tramite USB (UART)
- radio con antenna integrata
- transceiver RF compatibile con IEEE 802.15.4/ZigBee
- Banda RF: da 2.4 Ghz a 2.435 Ghz
- Transmit data rate: 250 kbps
- configuration EEPROM: 16 KB
- flash memory programmabile: 48 KB
- 1 MB di flash esterna per la registrazione dei dati (Measurement serial Flash)
- sensori opzionali (TPR2420)

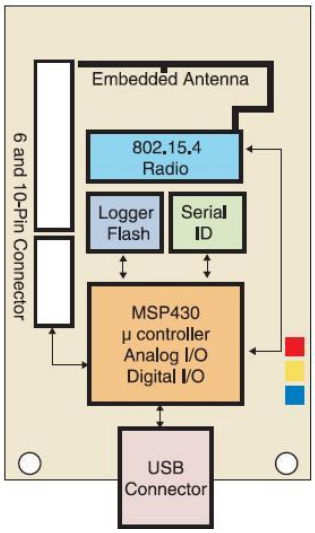


Figura 1.5 Telos rev. B block diagram

I due connettori d'espansione ed i jumper onboard possono essere configurati per controllare sensori analogici, periferiche digitali e display LCD.

# 2. L'adaptation layer 6LoWPAN

## 2.1 Introduzione

6LoWPAN [1] è stato disegnato per adeguare i pacchetti IPv6 al trasporto tramite IEEE 802.15.4 e rispondere alle seguenti esigenze, tipiche dei dispositivi in oggetto:

1. i dispositivi con limitata energia devono limitare il duty-cycle, mentre IP presuppone che i dispositivi siano sempre attivi.
2. le tecnologie wireless, come IEEE 802.15.4, non supportano multicast, cruciale per alcune caratteristiche di IPv6 ed il flooding consuma energia e banda.
3. le applicazioni dei dispositivi wireless beneficiano del multihop mesh networking, difficilmente integrabile con le normali soluzioni di routing IP.
4. le tecnologie radio utilizzate dai sensori wireless hanno una banda limitata e una frame di dimensioni ridotte, ad esempio IEEE 802.15.4 ha una frame di 127 byte, con payload disponibile nel range tra 72 e 116 byte mentre IPv6 prevede una frame di dimensioni minime di 1280 byte.
5. i protocolli standard non sono ottimizzati per reti wireless a bassa potenza, nelle quali può verificarsi la caduta di un nodo per failure o per esaurimento dell'energia.

La creazione di uno standard per il trasporto di IPv6 nelle reti di sensori wireless ha permesso di ridurre al minimo i prerequisiti e la riprogettazione di protocolli come il neighbor discovery (ND) ha permesso di utilizzare le caratteristiche peculiari di tali reti.
Alcuni concetti di 6LoWPAN sono strettamente legati alle funzione del link-layer, come ad esempio il PAN ID nella gestione dell'indirizzamento, ma il 6LoWPAN WG ha voluto mantenere un livello di generalità, evitando di utilizzare tutte le caratteristiche peculiari del livello MAC IEEE 802.15.4.
Le ridotte funzionalità richieste al MAC layer sottostante permetteranno di applicare 6LoWPAN in futuro ad un range di nuove tecnologie.

## 2.2 L'architettura 6LoWPAN

Assumendo per stub network una rete di dispositivi che riceve ed invia pacchetti ma che non agisce da rete di transito, l'architettura 6LoWPAN è realizzata da reti wireless a bassa potenza che realizzano IPv6 stub networks. L'interconnessione delle isole 6LoWPAN ad Internet realizza il Wireless Embedded Internet.
All'interno di ogni isola 6LoWPAN, i nodi condividono lo stesso IPv6 address prefix e sono possibili tre tipologie di rete LoWPAN: Ad hoc LoWPAN, Simple LoWPAN ed Extended LoWPAN.
Una rete Ad hoc LoWPAN non è connessa ad Internet ed opera senza infrastruttura, mentre le altre due tipologie di rete 6LowPAN sono connesse ad Internter tramite router detti edge.

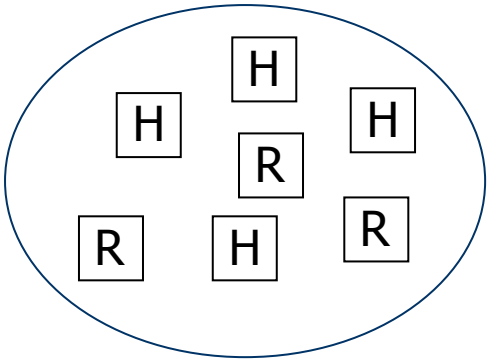


Figura 2.1 Ad hoc LOWPAN

Nel caso simple LoWPAN la rete è connessa tramite un solo router edge che a sua volta è connesso alla rete Internet tramite un backhaul link (point-to-point) oppure un backbone link verso un altro router.

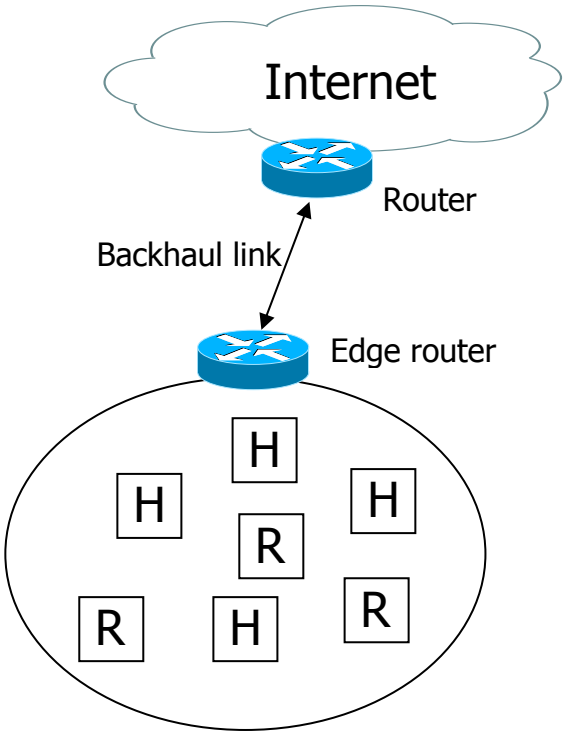


Figura 2.2 Simple LOWPAN

Nel caso Extended LoWPAN, sono presenti due router edge per l'interconnessione della rete ad Internet tramite backbone link (ad esempio ethernet).

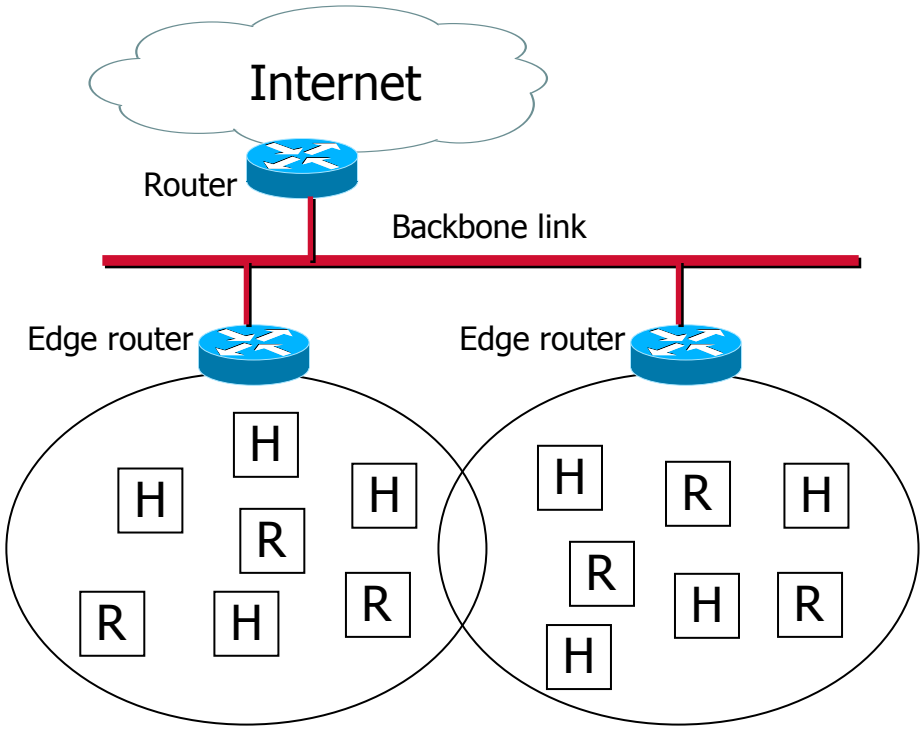


Figura 2.3 Extended LOWPAN

L'edge router è fondamentale nella gestione della compressione 6LoWPAN e del protocollo di Neighbor Discovery e nel caso si connetta la rete 6LoWPAN ad una rete IPv4, l'edge router esegue anche le operazioni di interconnettività.
All'interno dello stub, i nodi che assumono i ruoli di host o router sono liberi di muoversi all'interno del LoWPAN e generano le topology change, che possono essere causate anche da semplici variazioni delle condizione dei canali wireless.
Il multihop mesh topology può essere realizzato all'interno del 6LoWPAN tramite link-layer forwarding (Mesh-Under) oppure IP routing (Route-Over).
La comunicazione dei nodi LoWPAN e nodi IP esterni avviene in modo end-to-end, dato che ogni nodo è identificato tramite un indirizzo IPv6 univoco, tipicamente tramite ICMPv6 e UDP come trasporto per gli applicativi.

## 2.3 L'adaptation layer 6LoWPAN

L'adaptation layer deve adeguare i pacchetti IPv6 al trasporto da parte del sottostrato di livello 2 ed è necessario analizzare alcuni elementi che si riportano di seguito:

1. In un ambiente multi-access come IEEE 802.15.4 è necessario un indirizzamento link-layer per permettere di definire un destinatario del pacchetto, evitando che tutti i nodi debbano lavorare sui pacchetti in rete. Sulla base dell'indirizzo IP di destinazione l'adaptation layer deve ricavare l'indirizzo link-layer a cui consegnare il pacchetto.
2. LoWPAN non è connection-oriented e deve fornire le informazioni al successivo hop L2 per la gestione del pacchetto.

3. Il pacchetto IP deve essere incapsulato a livello 2 per consentirne il trasporto ed il destinatario L2 deve estrarlo. Alcune problematiche ne derivano:
    a. A differenza dell' IEEE 802.15.4, alcuni link-layer permettono di identificare la tipologia di pacchetto incapsulato, per questo 6LoWPAN contiene informazioni per decodificare il pacchetto incapsulato.
    b. I pacchetti IP potrebbero non essere contenuti nel layer 2 di trasporto, infatti IPv6 ha un MTU minimo di 1280 byte, mentre IEEE 802.15.4 può incapsulare pacchetti di una dimensione massima di 127 byte e perciò è necessario prevedere una forma di fragmentation.
    c. In LoWPAN, il tipico header IP/UDP occupa 48 byte, dimensione rilevante in relazione alla massima incapsulabile: per eliminare la ridondanza tra le informazioni di L3 e L2 è necessario l'header compression.

## 2.4 Link-layer

Il servizio di base richiesto al link-layer è l'invio di pacchetti di dimensioni limitate ad un altro nodo all'interno dell'area di copertura, senza necessità di affidabilità, infatti, 6LoWPAN è progettato per l'utilizzo in ambienti con ridotta energia, che non possono garantire l'affidabilità nella consegna del pacchetto.
A differenza delle reti cablate, in LoWPAN non è sempre definito il limite della copertura radio ed i nodi possono sperimentare una diversa raggiungibilità in base al nodo da raggiungere.
Per ogni nodo, l'insieme di nodi raggiungibili sono detti one-hop neighborhood ed il nodo può inviare broadcast locali che ogni nodo one-hop neighborhood può ricevere.
L'IEEE 802.15.4 MAC layer, definisce 4 tipologie di frame:

1. Data frame: utilizzate per il trasporto di dati, ad esempio le frame IPv6, sulla base delle specifiche 6LoWPAN.
2. Acknowledgement frame: vengono inviate dal ricevente immediatamente dopo la ricezione di una data frame, se l'acknowledgment bit è settato nella data frame MAC header.
3. Mac layer command frames: utilizzate per attivare svariati servizi a livello MAC, come association, dissociation da un coordinator e la gestione della trasmissioni sincronizzate
4. beacon frame: utilizzate dal coordinatore per strutturare la comunicazione con i propri associati.

6LoWPAN utilizza solo le data frame, che trasportano le protocol data unit (PDU) che a loro volta contengono pacchetti IPv6 o parti di essi.
6LoWPAN opera in modalità beaconless, utilizzando unslotted CSMA/CA e come prima operazione per la trasmissione, un nodo configura la variabile BE al valore macMin BE (default 3), poi attende un periodo casuale tra 0 e $(2^{BE}-1)$ unit times, dove unit time corrisponde a 20 symbol periods, quindi esegue un clear channel assessment: se il canale è libero, il nodo inizia la trasmissione, in caso contrario incrementa BE fino ad un valore macMAXBE (3-8, default 5) e continua ad attendere.
Un nodo interrompe il tentativo di trasmissione dopo aver tentato macMaxCSMABackoffs volte, un valore da 0 a 5, default 4.

Nel caso opzionale di acknowledgment il mittente può ripetere l'invio delle frame che non hanno ricevuto acknowledgment, per un numero di volte da 0 a 7, con default 3 e le frame vengo inviate con un ritardo minimo, dopo la ricezione, senza CSMA/CA.

## 2.5 L'indirizzamento link-layer

I nodi IEEE 802.15.4 si distinguono tramite identificativi EUI-64, composti da 8 byte ma, vista la limitata dimensione del pacchetto, è possibile ottenere un indirizzo corto di 16 bit.
Le data frame contengono sia l'indirizzo di source che quello di destination, in particolare quello di source è utilizzato dal nodo destinatario per gli aspetti di link-layer security e può essere rilevante in caso di mesh forwarding, inoltre IEEE 802.15.4 introduce un *PAN identifier* lungo 16 bit per separare nodi che appartengono a reti IEEE 802.15.4 con copertura in sovrapposizione.
In ogni caso l'indirizzamento link-layer non è sufficiente per identificare il nodo globalmente, non è gestibile tramite routing e non serve per determinare se un nodo appartiene allo stesso network.

## 2.6 Il formato 6LoWPAN

Il pacchetto IEEE 802.15.4 non contiene campi identificativi della tipologia del payload trasportato, perciò l'encapsulation 6LoWPAN deve supplire a questa mancanza.
6LoWPAN non utilizza un modello simile ai 2 byte di IEEE 802.3 Ethertype o l'header ad 8 byte SNAP, formati troppo estesi rispetto alla soluzione scelta che prevede che il primo byte del payload sia utilizzato come dispatch byte.

| **Range dispatch byte** | **Allocation** |
|---|---|
| **00** 000000 – **00** 111111 | Not a LoWPAN packet (NALP) |
| **01** 000000 | reserved for future use |
| **01** 000001 | IPv6 – uncompressed IPv6 packets |
| **01** 000010 | LOWPAN_HC1 – compressede IPv6 |
| **01** 000011 – **01** 001111 | reserved for future use |
| **01** 010000 | LOWPAN_BC0 – broadcast |
| **01** 010001 – **01** 011111 | reserved for future use |
| **01** 100000 – **01** 111110 | proposed for LOWPAN_IPHC |
| **01** 111111 | ESC – Additional Dispatch byte follows |
| **10** 000000 – **10** 111111 | MESH - Mesh header |
| **11** 000000 - **11** 000111 | FRAG1 - Fragmentation header (first) |
| **11** 001000 - **11** 011111 | reserved for future use |
| **11** 100000 – **11** 100111 | FRAGN – Fragmentation Header (sub) |
| **11** 101000 – **11** 101011 | proposed for fragment recovery |
| **11** 101100 – **11** 111111 | reserved for future use |

I primi 64 valori dei 256 possibili sono riservati, mentre i rimanenti 192 valori sono organizzati in 3 classi, che si identificano tramite i 2 bit iniziali. La tecnica di allocazione tramite il dispatch byte è in continua evoluzione e può essere verificata nel sito di IANA [11].

Il dispatch byte con valore 01000001 è definito per il trasporto dei pacchetti IPv6 non modificati, tipico dell'Ethernet.
In 6LowPAN è probabile che il mittente utilizzi qualche forma di compressione dell'header.
Il dispatch byte con valore valore 01010000, LOWPAN_BC0 permette il trasporto di una sequenza di numeri per il duplicate-detection in flooding-based broadcasting, mentre il valore 01111111=ESC è riservato per l'estensione del campo dispatch oltre un byte.
Alcuni formati definiti in 6LoWPAN si utilizzano per trasportare altre PDU 6LoWPAN nel payload e quando più header devono essere concatenate, devono rispettare il seguente ordine:

**Addressing:** mesh header, per il trasporto L2, contenente l'indirizzo di source e destination e l'hop count, seguito da una PDU 6LowPAN.

**Hop-by-hop processing:** header con opzione L2 hop-by-hop, come ad esempio un broadcast header, seguito da una PDU 6LoWPAN.

**Destination processing:** fragmentation header, per il trasporto dei fragment che devono essere riassemblati nel nodo destinatario, dopo il passaggio per L2 hop.

**Paylaod:** header che trasportano pacchetti L3, come IPv6 (01000001), LOW-PAN_HC1 (01000010) o LOWPAN_IPHC (011xxxxx).

## 2.7 Indirizzamento 6LoWPAN

Gli indirizzi coinvolti in 6LoWPAN sono 3: il Layer 3 IP e due indirizzi Layer 2, il 64-bit EUI-64 ed il 16-bit short address, dinamicamente assegnato.
L'indirizzamento Layer 2 EUI-64, globalmente unico, è composto da 24 bit assegnati al Vendor dall' OUI (organizationally unique identifier) e da 40 bit extension identifier.

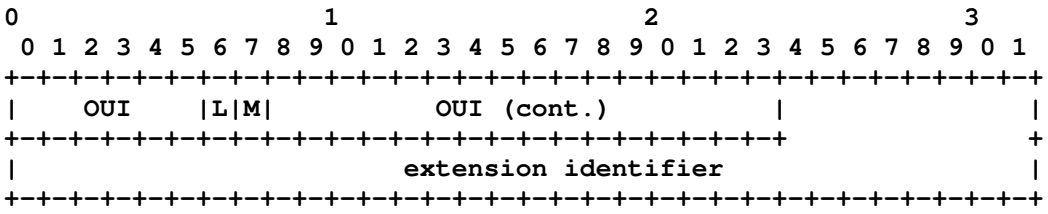


Figura 2.4 Indirizzamento EUI-64

All'interno dei primi 24 bit, due bit sono riservati: il bit M è utilizzato per distinguere gli indirizzamenti multicast dagli indirizzamenti unicast, mentre il bit L è utilizzato per distinguere gli indirizzi assegnati localmente dagli indirizzi assegnati globalmente dall'OUI.
A partire dall' EUI-64, l'indirizzo IPv6 si realizza unendo il prefisso di 64-bit con l'EUI-64, applicando la conversione del bit L (local-address) nel bit U (universal address).

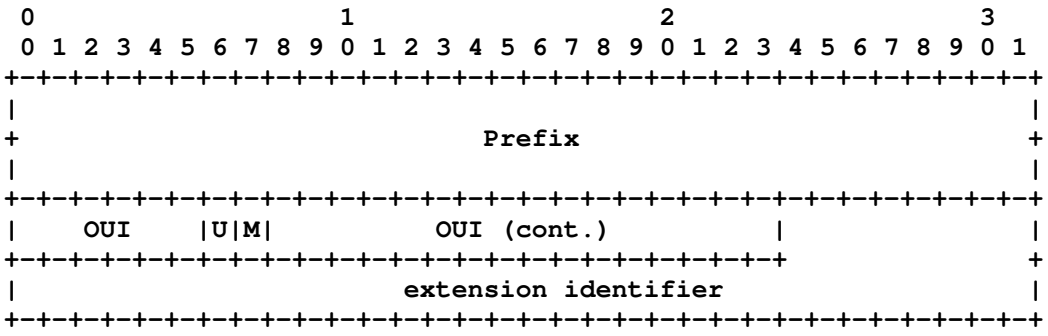


Figura 2.5 Indirizzamento IPv6 a partire da EUI-64

In 6LoWPAN la derivazione dell'indirizzo IPv6 dall'indirizzo link-layer è fondamentale per vari aspetti, tra i quali il compression header ed il protocollo di Neighbor Discovery: in particolare questa derivazione implica che IPv6 privacy extension for Stateless Address Autoconfiguration non è applicabile nel caso di 6LoWPAN, ma è compensata dalla possibilità di utilizzare l'indirizzamento a 16-bit, dinamicamente assegnato.

| range per i 16-bit del shot-address | assegnazione |
|---|---|
| 0xxxxxxxxxxxxxx | Disponibili per indirizzi unicast |
| 100xxxxxxxxxxxx | Indirizzi multicast |
| 1010000000000000 to 1111111111111101 | Riservato per 6LoWPAN |
| 111111111111111x | 0xFFFE e 0xFFFF riservati per IEEE 802.15.4 |

In IEEE 802.15.4 l'indirizzamento a 16-bit è assegnato dal PAN coordinator, invece in 6LoWPAN il gestore è l'edge router, in base alla protocollo di Neighbor Discovery. Per formare l'indirizzo IPv6 partendo dall'indirizzo a 16-bit assegnato dall'edge router, in base al Neighbor Discovery extension, il PAN Id è posizionato nei primi 16 bit. Nel caso il PAN Id non sia conosciuto, si possono utilizzare 16 bit a 0, continuando a rispettare la regola dello zero per l' universal bit.

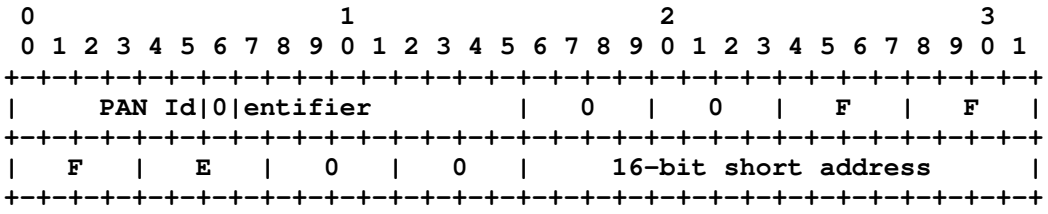


Figura 2.6 Indirizzamento IPv6 a partire da IEEE 802.15.4 16 bit

## 2.8 Forwarding e routing

In ogni nodo attraversato dal pacchetto 6LoWPAN si possono utilizzare due processi per la gestione del pacchetto: forwarding e routing.

Routing si basa sul *routing information base* (RIB) mentre forwarding si basa su *forwarding information base* (FIB). Il processo di routing alimenta la conoscenza del processo di forwarding, in modalità proattiva, in modo che si conosca la destinazione del pacchetto prima della ricezione dello stesso oppure in modalità reattiva, per permettere di colmare l'informazione di forwarding dopo l'arrivo del pacchetto.
In 6LoWPAN routing e forwarding possono essere effettuati a layer 3 ("Route-Over") oppure nei layer inferiori ("Mesh-Under") ed il forwarding è effettuato sulla stessa interfaccia dalla quale il nodo riceve il pacchetto.

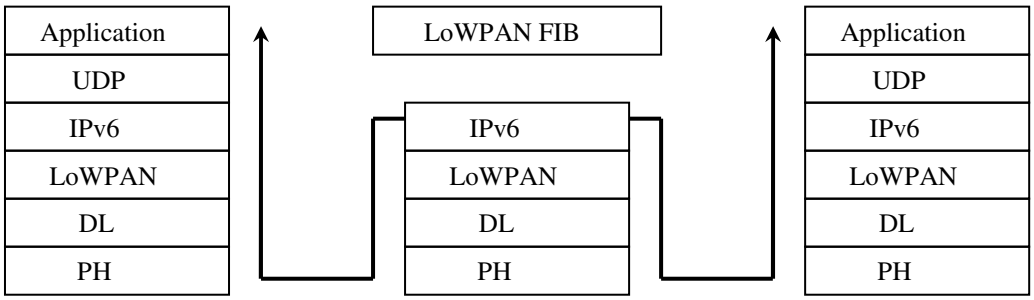


Figura 2.7 Routing in layer 3

Nel caso il routing venga effettuato a layer 3, non viene richiesto alcun supporto particolare al layer 2, infatti l'adaptation layer estrae il pacchetto layer 3. Inoltre, la frammentazione e ricomposizione del pacchetto vengono eseguiti in ogni hop, dato che l'indirizzo layer 3 è nella prima parte del pacchetto.
Nel caso routing e forwarding vengano effettuati a layer 2, il processo decisionale si basa sull' indirizzamento a 64 bit EUI-64 oppure sull'indirizzamento a 16-bit e può essere trasparente all'adaptation layer LoWPAN.

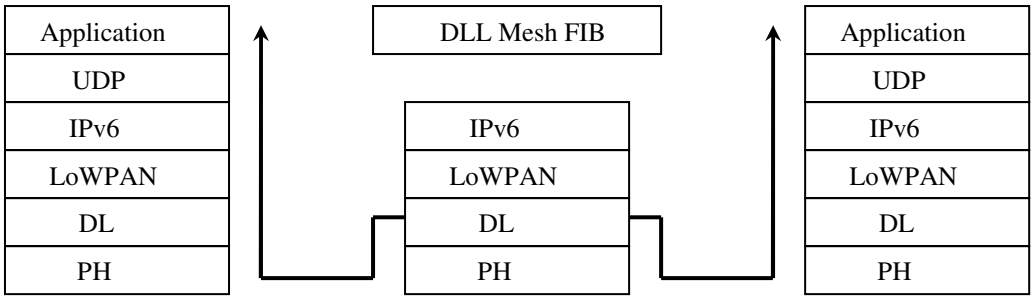


Figura 2.8 Routing in layer 2

Nel caso routing e forwarding vengano effettuati a layer 2, coinvolgendo l'adaptation layer LoWPAN, il nodo deve conoscere l'indirizzo layer 2 di origine e di destinazione, mentre nel pacchetto gestito compaiono gli indirizzi di origine e destinazione dell'hop locale. Per questo in 6LoWPAN è definito il mesh header.

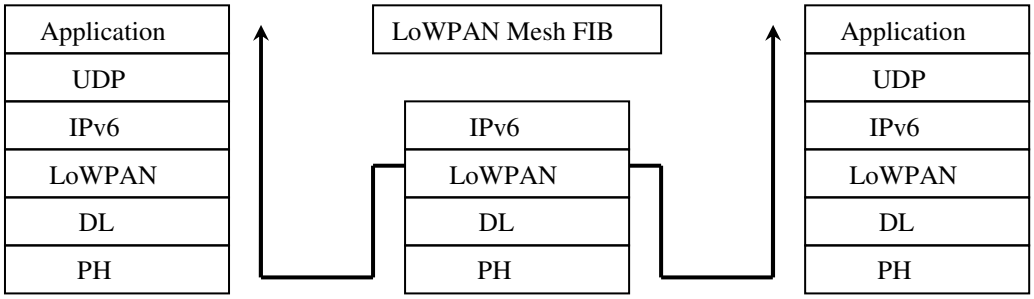


Figura 2.9 Routing in layer 2 con adaptation layer

Oltre agli indirizzi di origine e destinazione, nel mesh header è presente un campo *hop left* di 4 bit, estendibile, che viene decrementato prima del forwarding e se viene raggiunto il valore nullo, il pacchetto viene scartato.
Inoltre, nel mesh header sono presenti due campi, V ed F che indicano se l'indirizzamento è 16-bit o 64-bit EUI-64.
Di seguito il mesh header nel caso standard e nel caso di hop left esteso:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|V|F|HopsLft| originator address, final address ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0|V|F|1 1 1 1|   Hops Left   |originator address, final address...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\_  dispatch  _/
```

Figura 2.10 Mesh header

## 2.9 Header compression

La limitata dimensione del pacchetto dell' IEEE 802.15.4 e la lunghezza dell'header IPv6 impongono la compressione di alcune parti del pacchetto, al fine di evitare la frammentazione, che è naturalmente inefficiente.
Nel caso la lunghezza del pacchetto IPv6 permetta di evitare la frammentazione, la lunghezza del pacchetto incide sul consumo energetico del sensore e giustifica comunque la compressione.
La compressione può essere applicata se è presente una ridondanza dell'informazione, a livello di singolo pacchetto o in una sequenza di pacchetti.
Tra le tecniche sviluppate troviamo l'header compression e il data compression, che restituisce i migliori risultati se applicata nell' application layer.
Data compression è sensibile alla perdita di pacchetti che possono contenere informazioni per la decodifica dell'intera sequenza di pacchetti, mentre header compression può essere applicata in modalità end-to-end oppure hop-by-hop.
Nel caso header compression si applichi end-to-end, si deve limitare alla compressione degli header all'interno dell' IP, dato che i router tra l'origine e la destinazione devono poter vedere l'IP header, mentre nel caso header compression si

applichi in modalità hop-by-hop è possibile comprimere in modo più efficace l'header, compreso l'IP header.
La compressione dell'header in modalità hop-by-hop viene effettuata prima dell'invio nel link e la decompressione alla ricezione del pacchetto dal link, applicando tecniche diverse ad ogni hop, spostando le decisione in merito all' header compression a livello locale tra due nodi confinanti.
Gli algoritmi di header-compression possono essere flow-based oppure stateless: nel caso flow-based è necessaria la gestione dell'informazione di stato relativa al flusso, detta context e si deve prevedere una protezione nel caso di perdita di pacchetti, rendendo gli algoritmi molto complessi (i.e. Van Jacobson header compression, ROHC).
In un ambiente caratterizzato dalla limitazione delle risorse come LoWPAN è consigliabile evitare algoritmi complessi e per questo la versione originale di 6LoWPAN utilizza la compressione stateless.
La compressione stateless 6LoWPAN comunque manifesta una limitata applicabilità ed il 6LoWPAN Working-Group sta sviluppando un nuovo metodo di header compression context-based ormai giunto alla versione draft-ietf-6lowpan-hc-06.
Di seguito viene riportato l'header IPv6, secondo RFC 2460, che sarà soggetto ai diversi tipi di compressione:

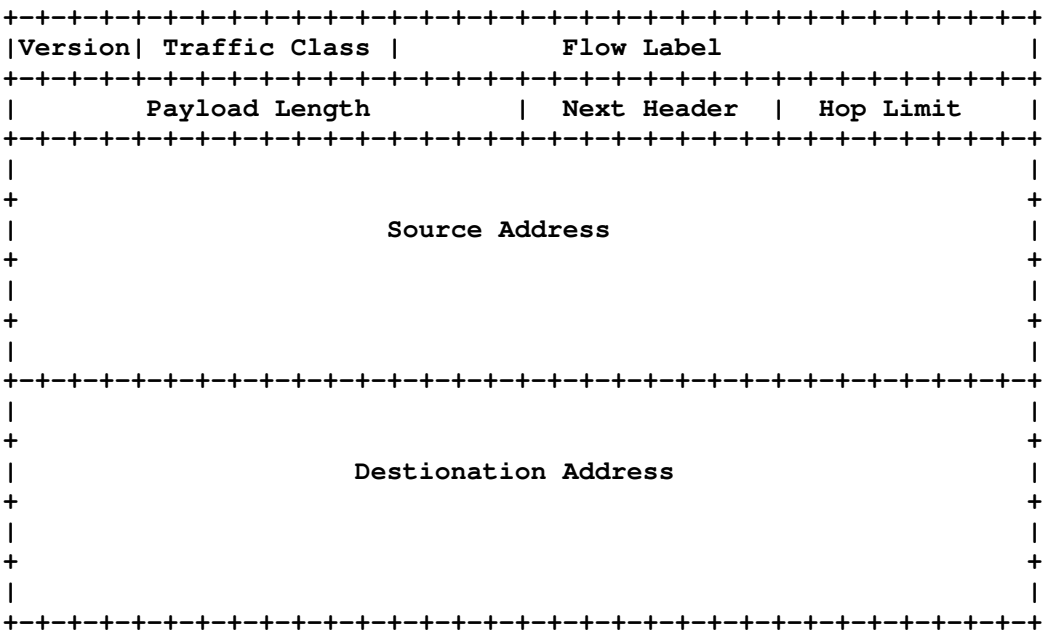


Figura 2.11 Header IPv6

### 2.9.1 Stateless header compression

La versione originale di 6LoWPAN prevede due forme di stateless header compression che possono coesistere: HC1 per la compressione dell' header IPv6 (dispatch 01000010) e HC2 per la compressione dell'UDP header, che ovviamente non richiedono alcuna fase di agreement tra i nodi precedente alla trasmissione.
In particolare nel caso di HC1, l'ultimo bit del byte successivo al dispatch indica l'eventuale presenza di HC2, i cui parametri sono definiti nel byte successivo.
Di seguito i byte iniziali nel caso di HC1 ed nel caso di HC1 con HC2:

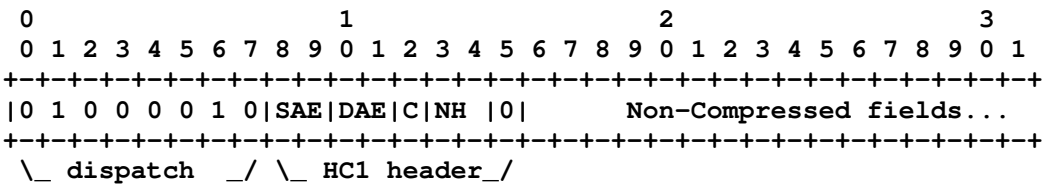


Figura 2.12 HC1

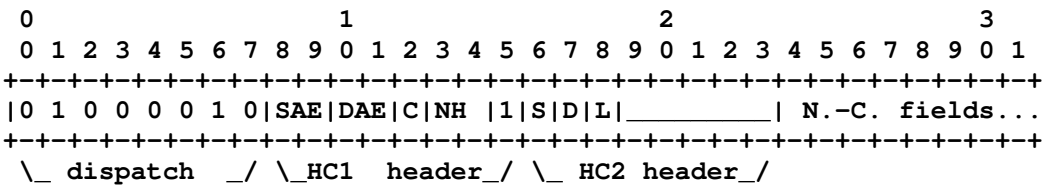


Figura 2.13 HC1 con HC2

La compressione stateless HC1/HC2 sfrutta la ridondanza delle informazioni all'interno del pacchetto, in particolare il legame tra l'indirizzo IP e l'indirizzo layer 2. La parte IID dell'indirizzo IP di source e destination è creata a partire dal MAC address, perciò è ridondante e può essere rimossa. I primi 64 bit dell'indirizzo IP possono essere ottimizzati solo nel caso di link-local address (FE80::/64). La compressione delle due metà dell'indirizzo di origine e destinazione è segnalata tramite i 2 bit del campo SAE e DAE, come riportato nella seguente tabella.

| **SAE e DAE** | **Prefix** | **IID** |
|---|---|---|
| 00 | nessuna compressione | nessuna compressione |
| 01 | nessuna compressione | rimosso e ricavato dall'indirizzo layer 2 o mesh address |
| 10 | rimosso e considerato link-local (FE80::/64) | nessuna compressione |
| 11 | rimosso e considerato link-local (FE80::/64) | rimosso e ricavato dall'indirizzo layer 2 o mesh address |

La parte rimanente dell' header HC1 riguarda la compressione di parti dell' header IPv6 non legate all'indirizzamento:

- la versione è comunque 6 e perciò può essere rimossa.
- i campi trafffic class e flow label dell' header IPv6 sono spesso nulli ed in tal caso si porta ad 1 il bit C dell' header HC1, mentre se il bit C è nullo i due campi sono riportati nella parte non compressa dell'header.
- la lunghezza del payload può essere ricavata dalla lunghezza del PDU 6LoWPAN.
- il next header dell' header IPv6 è un byte, ma assume più frequentemente un range limitato di valori, riportati in tabella
- Hop limit non è comprimibile e perciò è inviato senza alcuna compressione.

Di seguito la tabella con le specifche del campo HC1 NH:

| valori del campo HC1 NH | |
|---|---|
| 00 | Next header senza compressioni |
| 01 | Next header = 17, UDP |
| 10 | Next header = 1, ICMP |
| 11 | Next header = 6, TCP |

La presenza dell'header HC2 è indicata dell'ultimo bit dell'header HC1 e il corrispondente valore 01 di HC1 NH per UDP.
I campi dell'header HC2 indicano la compressione della source port, destination port e length, rispettivamente bit S, D e L.
Il campo length può essere ricavato dalla lunghezza dei byte del payload rimanenti, mentre la compressione della porta di source e destination è più complessa e richiede l'identificazione di uno spazio preferenziale che può essere compresso.
In particolare 6LoWPAN identifica le porte comprese tra 61616 (0xF0B0) e 61631 (0xF0BF), che possono essere rappresentate tramite 4 bit e che permettono di risparmiare 3 byte se entrambi i bit S e D sono settati ad 1.
Il campo UDP checksum non è mai compresso.
I campi non compressi seguono gli header HC1 o HC1/HC2 ed iniziano sempre con Hop Limit secondo le indicazioni seguenti:

- source address prefix (64 bit), se il primo bit di SAE è nullo
- source address interface identifier (64 bit), se secondo bit di SAE è nullo
- destination address prefix (64 bit), se primo bit di DAE è nullo
- destination address interface identifier (64 bit), se secondo bit di DAE è nullo
- traffic class (8 bit) e flow label (20 bit), se il bit C è nullo
- next header (8 bit), se NH è zero
- campi non compressi da HC2
- next header e payload non soggetti a compressione

La specifica del formato 6LoWPAN non indica come gestire i campi di lunghezza non multipla di 8 come ad esempio flow label (20 bit).

### 2.9.2 Context-based header compression HUI HC-00

La maggior parte dei pacchetti che vengono scambiati in una rete LoWPAN coinvolge nodi esterni alla rete, richiedendo l'utilizzo di indirizzi globali e perciò solo 8 dei 32 byte sono comprimibili.
La prima versione del draft-hui-6lowpan-hc [4] si basa su contesti condivisi tra i nodi del network 6lowpan, per comprimere l'indirizzamento IPv6 elidendo i bit del prefix.
Come nel caso stateless, la compressione dell' header context-based prevede la compressione dell'header IP (LOWPAN_IPHC) e la compressione opzionale del next header (LOWPAN_NHC).
Questo draft contiene un metodo per la compressione degli indirizzi multicast più comuni ed un framework per la compressione del next header e dell'header UDP.
Per quanto riguarda la compressione dell'header IPv6, il campo version ha sempre il valore 6, traffic class e flow label sono entrambi di valore 0, la lunghezza del payload

si può ricavare dai layer più bassi, dal 6lowpan frag header o dall' 802.15.4 header, l'hop limit è solitamente impostato ad un valore comune.
Inoltre, gli indirizzi si formano utilizzando il link-local prefix o un solo prefisso globale nell'intero network e l'IID è derivato dall'indirizzamento link-layer esteso a 64-bit oppure corto a 16-bit.
La codifica LOWPAN_IPHC utilizza un solo otteto in aggiunta al byte di dispatch, con i campi non compressi a seguire e può comprimere l'header IPv6 fino ad un singolo otteto, nelle comunicazioni link-local mentre nelle comunicazioni multi-hop LOWPAN_IPHC può comprimere l'header IPv6 fino a 6 otteti, composto da un otteto per LOWPAN_IPHC, un otteto per Hop Limit, 2 otteti per il source address e 2 otteti per il destination address.
Di seguito viene riportato il formato di codifica LOWPAN_IPHC:

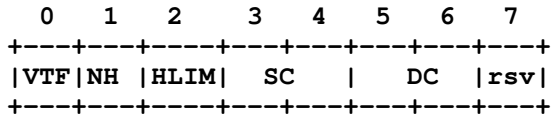


Figura 2.14 LOWPAN_IPHC secondo HC-00

Di seguito le specifiche dei campi della codifica LOWPAN_IPHC

| **VFT** | **version, traffic e flow label** |
|---|---|
| 0 | trasportati nell'header |
| 1 | rimossi dall'header |
| **NH** | **next header** |
| 0 | 8 bit trasportati nell'header |
| 1 | 8 bit rimossi e compressi con LOWPAN_NHC |
| **HLIM** | **hop limit** |
| 0 | 8 bit trasportati nell'header |
| 1 | 8 bit rimossi e se il destination address non è assegnato all'interfaccia, hop limit assume valore 64, se è assegnato all'interfaccio hop limit assume valore 1 |
| **SC e DC** | **compressione della source e destination address** |
| 00 | 128 bit trasportati nell'header |
| 01 | 64 bit trasportati nell'header (IID) |
| 10 | 16 bit trasportati nell'header |
| 11 | indirizzo interamente rimosso dall'header |
| **rsv** | **riservato** |

In particolare il byte di dispatch concorre nella compressione della parte prefix dell'indirizzo unicast IPv6, in caso di compressione a 64, 16 o completa: il valore del dispatch 0x03 indica un prefix link-local, mentre il valore 0x04 indica un prefix globale condiviso tra i nodi.
Questi valori di dispatch ricadono negli intervalli riservati alle codifiche che non appartengono a 6lowpan (NALP, intervallo 0x00-0x63) nei draft più recenti.
Nel caso il prefix globale non sia univoco si genera un errore nella fase di integrity check.

Per quanto riguarda la parte IID, nel caso di compressione a 16 bit (codifica 10), gli ultimi 16 bit sono trasportati nell'header, ma devono avere il bit più significativo posto a 0, per differenziarsi dalla compressione multicast che utilizza la modalità 16 bit.
Nel caso l'indirizzamento venga completamente rimosso (codifica 11), la parte IID si ricava dall'indirizzamento del link layer con gli altri bit a 0, anche se RFC 4944 [3] richiede l'inserimento del PAN ID.
L'indirizzamento IPv6 multicast può essere compresso a 16 bit, utilizzando i range di indirizzi e di seguito viene riportato la codifica per l'indirizzamento multicast:

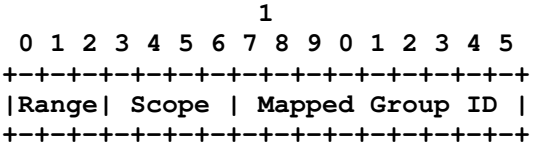


Figura 2.15 Compressione Multicast

Di seguito le specifiche dei campi della codifica dell'indirizzamento IPv6 multicast:

| | |
|---|---|
| Range, bit 0-2 | range per la compressione 6lowpan dell'indirizzamento multicast |
| Scope, bit 3-6 | bit di scopo multicast (secondo RFC 4007) |
| Mapped group ID | identificativo per la mappatura dei gruppi multicast, seocndo specifiche IANA |

Come visto in precedenza il next header viene rimosso con il bit NH della codifica LOWPAN_IPHC e in questo caso viene utilizzata la codifica LOWPAN_NHC per il next header:

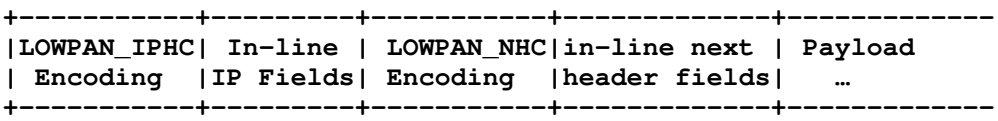


Figura 2.16 LOWPAN_IPHC con LOWPAN_NHC

La compressione LOWPAN_NHC si basa sul valore di un unico bit ID, secondo la seguente compressione dell'header LOWPAN_UDP:

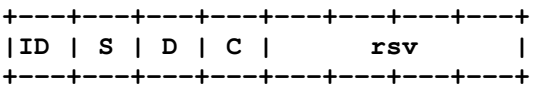


Figura 2.17 LOWPAN_NHC

Di seguito le specifiche dei campi della codifica LOWPAN_UDP:

| **ID** | **tipologia del next header** |
|---|---|
| 0 | IPv6 next header 17 and LOWPAN_UDP compression format |
| 1 | IPv6 next header diverso da 17 e non definito |
| **S** | **Source port** |

| | |
|---|---|
| 0 | 16 bit trasportati nell'header |
| 1 | primi 12 bit rimossi, ultimi 4 bit trasportati |
| **D** | **Destination port** |
| 0 | 16 bit trasportati nell'header |
| 1 | primi 12 bit rimossi, ultimi 4 bit trasportati |
| **C** | **checksum** |
| 0 | 16 bit trasportanti nell'header |
| 1 | 16 bit rimossi |

Di seguito viene riportato l'analisi di un tipica frame con compressione HC-00 e LOWPAN_UDP:

frame IEEE 802.15.4:
02 24 61 88 1B 22 00 17 67 05 00 03 D4 F0 00 05 67 17 E0 01 20 08 6D 73 67 20 72 65 63 65 69 76 65 64 21 00 00 EB

paylod IEEE 802.15.4: 03 D4 F0 00 05 67 17 E0 01 20 08 6D 73 67 20 72 65 63 65 69 76 65 64 21 00

dispatch: 0x03 local-link

codifica LOWPAN_IPHC: 0xD4, 11010100

VFT=1, versione traffic e flow label rimossi
NH=1, next header rimosso, compressione LOWPAN_NHC
HLIM=0, hop limit trasportato nell'header
SC=10, source address compresso a 16 bit
DC=10, destination address compresso a 16 bit

hop limit, F0

16 bit source address: 00 05
16 bit destination address: 67 17

E0 01, 1110000 00000001, LOWPAN_UDP
ID=1, compressione LOWPAN_UDP, ma standard richiederebbe ID=0
S=1, source port compressa
D=1, destination port compressa
C=0, checksum trasportato
rsv=0000, reserved

source e destinazion port : 01, source port F0B0, destination port F0B1

checksum: 20 08

payload UDP: 6D 73 67 20 72 65 63 65 69 76 65 64 21 00 (msg received!)

### 2.9.3 Context-based header compression IETF HC-04

La compressione degli indirizzi globali richiede la creazione di un valore di context all'accesso della rete LoWPAN, valore che può essere acquisito tramite Neighbor Discovery (6LoWPAN-ND).
Il context deve essere sincronizzato tra i nodi di origine e di destinazione, che eseguono la compressione e decompressione dell'header, inoltre i layer superiori devono garantire un meccanismo di protezione, tramite un checksum dell'header o altre forme di verifica dell'integrità a livello applicativo.

La compressione dell'header IP necessita di 13 bit e per realizzare una compressione efficace, si utilizzano 5 bit del dispatch ed il byte successivo. Inoltre può essere utilizzato il terzo byte per la definizione di un context per l'indirizzo di source e destination diverso dal default, indicato dal bit C, definito come bit CID nelle specifiche.

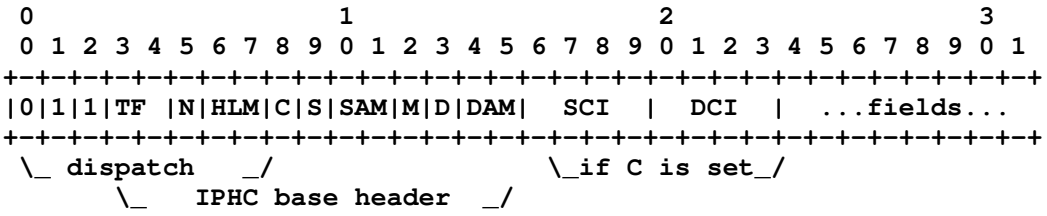


Figura 2.18 LOWPAN_IPHC secondo HC-04

L'header LOWPAN_IPHC consiste di flag che controllano quali campi dell'header IPv6 sono compressi:

- i 2 bit TF controllano come vengono compressi i 6 bit del campo traffic class ed i 20 bit del campo flow label, che vengono eliminati completamente solo se TF=11, mentre negli altri casi si riportano sempre almeno i 2 bit ECN.
- il bit N controlla se il next header è inviato senza variazioni o se utilizza LOWPAN_NHC.
- per comprimere senza eliminare il campo Hop Limit, si utilizzano 3 valori predefiniti (1, 64 e 255) selezionabili tramite 2 bit, che se assumono valore 0 indicano la presenza di un byte non compresso che ne indica il valore reale.
- La compressione dell'indirizzo di origine e destinazione sono controllati dai campi S/SAM e D/DAM, in particolare i campi S e D indicano se la compressione dell'indirizzamento è context-based mentre i campi SAM e DAM indicano il numero di bit che sono trasmessi non compressi, che combinati con gli altri campi dell'IPv6 header permettono di ricostruire l'indirizzamento IPv6.
- Il campo M indica se la destinazione è un indirizzo multicast.

Di seguito si riportano le indicazioni sui valori S/SAM e D/DAM nel caso M=0.

| S/SAM | Inline | Descrizione |
|---|---|---|
| 0 00 | 128 | inline (indirizzamento non compresso) |
| 0 01 | 64 | FE80:0:0:0:inline (link-local + 64-bit Interface ID) |
| 0 10 | 16 | FE80:0:0:0:0:0:inline (link-local + 16-bit Interface ID) |
| 0 11 | 0 | FE80:0:0:0:link-layer (link-local + link-layer source address) |
| 1 00 | -- | riservato |
| 1 01 | 64 | context[0..63]:inline (context + 64-bit Interface ID) |
| 1 10 | 16 | context[0..111]:inline (context + 16-bit Interface ID) |
| 1 11 | 0 | context[0..127] (context) |

Nell'header IPv6 il campo version non viene riportato, come anche il campo IPv6 payload length, che è legato alla lunghezza del PDU 6LowPAN, che può essere ricavato direttamente dall'header IEEE 802.15.4 o dal 6LowPAN fragmentation header nel caso di frammentazione.

Dopo i 2 o 3 byte inziali, seguono tutti i campi dell'header IPv6 che non sono stati compressi, nello stesso ordine in cui compaiono nell'header IPv6 non compresso. A seguire il next header, ad esempio UDP che, se il bit N del dispatch è settato, è compresso utilizzando LOWPAN_NHC.
LOWPAN_NHC è definto con un metodo estensibile, simile al dispatch byte ed attualmente è utilizzato per la compressione dell'UDP o per la compressione di IPv6 extension headers, di seguito i base header.

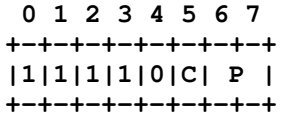


Figura 2.19 LOWPAN_NHC per UDP

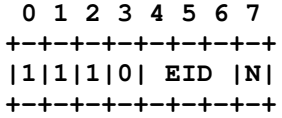


Figura 2.20 LOWPAN_NHC per IPv6 extension header

Nel caso LOWPAN_NHC per UDP il bit C ed il campo P indicano rispettivamente se l'UDP checksum è riportato e se i campi di source e destination di UDP sono compressi.
Ovviamente la rimozione del checksum non è consigliabile ed è configurabile solo se c'è un meccanismo di controllo della sessione, compreso l'indirizzamento. La rimozione del checksum deve essere autorizzata dalla source del pacchetto che può prevedere il danno che può derivare dalla corruzione del pacchetto.

## 2.10 Fragmentation

IPv6 non prevede campi per la frammentazione e non prevede frammentazione durante il percorso: se la source del pacchetto vuole utilizzare la frammentazione, inserisce un header per la frammentazione con extension header.
In IPv6 il minimum MTU è stato portato a 1280 byte (68 byte in IPv4), permettendo l'inserimento di tunnelling header in Ethernet ed è definito un minimum reassembly size di 1500 byte (576 in IPv4). Per i link layer che non riescono a rispettare il minimum MTU è necessario prevedere l'adeguamento a livello di adaptation layer.
6LoWPAN può far affidamento sull' error-checking fornito dal link layer e sulla possibilità di inviare frame di lunghezza variabile ma non può far affidamento sulla consegna ordinata delle frame.
6LoWPAN non prevede un more-fragments flag, ma copia la dimensione del pacchetto da ricostruire in ogni frammento, tramite gli 11 bit di *datagram_size* e questo permette al nodo che riceve i frammenti, di allocare un buffer per l'intero pacchetto dopo la ricezione di un qualsiasi frammento. Come in altri casi già visti, i primi 3 bit del datagram_size sono condivisi con gli ultimi 3 bit dal dispatch byte. Oltre al sender link layer, al destination link layer ed al datagram_size, 6LoWPAN prevede il campo datagram_tag di 16 bit per distinguere i pacchetti da riassemblare. Inoltre è presente il campo datagram_offset di 8 bit, che indica la posizione del frammento all'interno del pacchetto ricostruito, con unità da 8 byte.

Di seguito il formato del frammento iniziale:

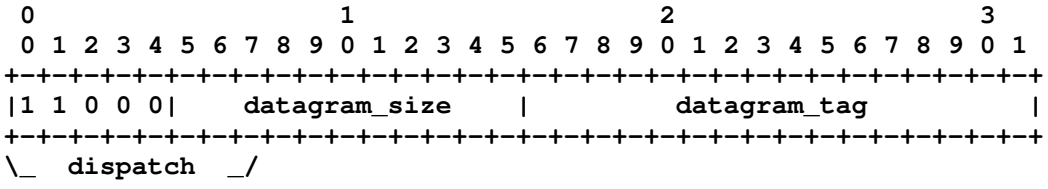


Figura 2.21 Frammento iniziale

Di seguito il formato del frammento non iniziale:

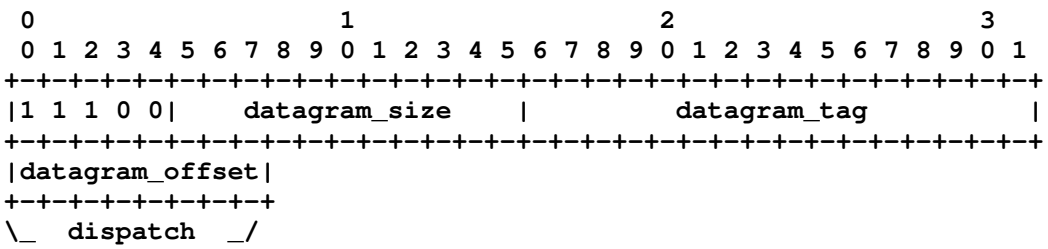


Figura 2.22 Frammento non iniziale

Se il primo frammento utilizzasse il formato con il datagram_offset, avrebbe 8 bit a zero, che possono essere rimossi, con un risparmio nel primo frammento, utilizzando però un valore di dispatch diverso per il primo frammento.
Di seguito viene riportata la procedura necessaria per la frammentazione quando un nodo deve inviare una 6LoWPAN PDU troppo grande per rientrare in una frame del link layer:

1- si pone nella variabile *packet_size* il valore della dimensione del pacchetto IPv6 e nella variabile *header_size* il valore dell'header 6LoWPAN che si dovrebbe inviare nel caso di una single frame. Nel caso parte dell'header o del payload IPv6 siano state ridotte dalla fase di compressione 6LoWPAN, si dovrà adeguare il valore di *header_size* ed in sintesi la somma di *header_size* e *packet_size* restituiscono la dimensione della 6LoWPAN PDU, che si dovrebbe inviare nel caso non servisse frammentazione.
2- si pone nella variabile *max_frame* il valore dello spazio che avanza nella frame link-layer, dopo aver previsto gli header PHY, MAC, address e security, i trailers ed eventuali headers 6LoWPAN che devono essere inseriti prima del pacchetto (ad esempio mesh headers nel caso di Mesh-Under). La frammentazione è necessaria solo nel caso *max_frame* < *header_size* + *packet_size*. Si noti che *max_frame* può dipendere da next-hop destination, impostazioni di security e dalla dimensione dell'indirizzamento.
3- viene incrementato il valore della variabile globale *datagram_tag* e tale valore verrà utilizzato in tutte le frame di questa PDU.
4- nell'ottica di inviare dati del pacchetto IPv6 nel primo frammento che rispettino lo spazio a disposizione ma che siano multipli di 8 byte, si calcola *max_frag_initial* = DIV[(*max_frame*-4-*header_size*)/8]*8, dove si evidenziano 4 byte per lo spazio necessario all'header del frammento iniziale.

5- Si spediscono i byte *max_frag_initial* + *header_size* in un frammento iniziale, inserendo in testa i 4 byte dell'initial fragment header.
6- si pone nella variabile *position* il valore *max_frag_initial*.
7- si pone nella variabile max_frag il valore DIV[(max_frame-5)/8]*8, considerando 5 byte per l'header del frammento non-iniziale.
8- Finchè *packet_size-position* > *max_frame*-5, si inviano i prossimi *max_frag* byte in un frammento non iniziale, anteponendo l'header non iniziale di 5 byte, si aggiorna *datagram_offset* al valore *position*/8 e si incrementa *position* di *max_frag*.
9- Si inviano i rimanenti byte in un frammento non iniziale, inserendo in *datagram_offset* il valore *position*/8.

Di seguito viene riportata la procedura necessaria per la ricostruzione di una PDU a partire dai frammenti:

1- si costituisce la combinazione dei seguenti valori, che cambiano per ogni operazione di frammentazione di una PDU: l'indirizzo di *source*, *destination*, *datagram_size* e *datagram_tag*.
2- se non è stato allocato un buffer per la ricostruzione della combinazione del punto 1, si crea il buffer di dimensione pari a *datagram_size* e si inizializza una lista vuota dei frammenti ricevuti.
3- per il frammento iniziale si pone a zero il valore di *datagram_offest*, si rimuovono i 4 byte dell'header del frammento, si eseguono le operazioni di decodifica ed espansione e si calcolano i campi omessi, si pone in *data* il contenuto ed in *frag_size* la dimensione del pacchetto estratto.
4- per un frammento non iniziale, si pone nella variabile *datagram_offest* il valore del relativo campo, si rimuovono i 5 byte dell'header, si pone in *data* il contenuto del frammento ed in *frag_size* la dimensione del frammento ricevuto decrementato di 5 byte.
5- si pone nella variabile *byte_offset* il valore *datagram_offset**8.
6- si verifica che il valore *frag_size* sia un multiplo di 8 oppure *byte_offset*+*frag_size*=*datagram_size*, per il frammento finale: in caso contrario l'operazione non avrà esito positivo.
7- se uno dei valori della lista dei frammenti ricevuto si sovrappone in modo non identico all'intervallo [*byte_offeset*, *byte_offset*+*frag_size*], l'operazione non avrà esito positivo.
8- si incrementa la lista dei frammenti con l'intervallo [*byte_offeset*, *byte_offset*+*frag_size*].
9- si copia il contenuto di *data* nel buffer, nelle posizioni che iniziano da *byte_offset*.
10- Se la lista degli intervalli ricopre tutto lo spazio, la ricomposizione è completa ed il buffer contiene un pacchetto IPv6 di dimensione *datagram_size* ed ora si possono eseguire le operazioni che richiedono l'intero pacchetto, come ad esempio UDP checksum.

In caso di fallimento dell'operazione è possibile un nuovo tentativo di ricomposizione con un nuovo buffer.
La frammentazione comunque è un operazione da evitare in un ambiente con risorse limitate come il WSN, poichè l' incertezza del processo di ricezione dei frammenti da ricomporre rende difficoltosa la gestione delle risorse assegnate, inoltre senza un

meccanismo di frame acknoledge, il disaccoppiamento tra i singoli frammenti ed il PDU, può aumentare l'inefficienza di rete, nel caso di perdita dei frammenti che costringono la ritrasmissione dell'intera PDU.
Inoltre, la variabilità dei formati e dei paramentri utilizzati rende incerta la determinazione di uno spazio minimo disponibile per tutte le frame link-layer (*max_frame*), che può differire anche tra nodi contigui.
Per una stima ragionevole di *max_frame* che eviti la frammentazione si parte da 127 byte di IEEE 802.15.4, si tolgono i seguenti:

1. 5 byte per headers e trailers.
2. da 6 a 18 byte per l'indirizzamento.
3. 30 byte per ottenere un livello di security elevato, 20 byte per livello intermedio.
4. il nuovo compression header può comprimere gli header IPv6 e UDP ad 11 byte, ma necessita di 20 o più byte.

Perciò un'applicazione UDP dovrà utilizzare payload tra 50 e 60 byte per avere elevata probabilità di invio del PDU senza frammentazione lungo tutto il percorso.

## 2.11 Multicast

IPv6 considera il multicasting come elemento basilare ed è stato inserito da subito nelle specifiche, rimpiazzando il meccanismo di broadcasting della subnet, mentre il link-layer IEEE 802.15.4 non prevede meccanismi di multicast, ma solo un indirizzo di broadcast che permette di raggiungere tutti i nodi.
6LoWPAN supporta il routing protocol mesh-under che prevede funzionalità di multicast in aggiunta al metodo scontato del flooding fornito da IEEE 802.15.4, con sequence number.
6LoWPAN implementa multicast a livello 2 in mesh-under routing riservando primi 13 bit del 16-bit short address per la traduzione degli indirizzi multicast IPv6, secondo varie metodologie.

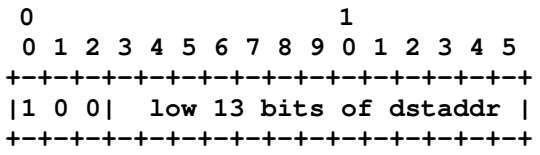


Figura 2.23 Multicast in mesh-under

## 2.12 Bootstrapping

Quando un WNS in una rete 6LoWPAN viene acceso, deve eseguire le seguenti operazioni:

- trovare il LoWPAN a cui andrà a far parte.
- definire i parametri di networking, come il prefisso dell'IP address ed il proprio indirizzo IPv6.
- definire le associazione di security con le entità del network.

- definire i PATH verso le entità rilevanti del network, gestire i PATH e possibilmente effettuare forwarding per gli altri nodi.
- stabilire i paramentri del livello applicativo.
- stabilire le associazioni di security con le entità rilevanti a livello applicativo.
- far partire i protocolli del livello applicativo.

Alcune operazioni tra quelle elencate devono essere ripetute frequentemente nel tempo, mentre altri parametri sono più statici.
Il setup dei parametri può essere organizzato in due fasi:

- Commissioning, questa fase richiede l'intervento umano per definire alcuni parametri iniziali, comprende le fasi di installazione, configurazione e provisioning. Più la fase di commissioning è rigida e meno dinamica sarà la rete di sensori, utilizzando peculiarità della successiva fase di bootstrapping. La fase di commissionig può essere realizzata durante la produzione, oppure durante l'installazione (ad esempio tramite USB), oppure il sensore può essere configurato utilizzando la stessa rete 6LoWPAN, con tutte le complicazioni connesse.
- Bootstrapping, dopo il commissioning, il nodo è in grado di funzionare autonomamente, un protocollo che è sicuramente coinvolto in questa fase è Neighbor Discovery.

Il sensore può essere in passive-mode, in attesa della configurazione base, oppure in active mode, in attesa di entrare nel network, in base ai parametri preconfigurati, per ottenere dal network i parametri di configurazione rimanenti.
Un parametro deve essere sicuramente impostato in ogni sensore secondo lo standard 6LoWPAN: il codice EUI-64 che viene utilizzato come indirizzo MAC a 64-bit.

### 2.12.1 Neighbor Discovery

IPv6 Neighbor Discovery è un elemento fondamentale nella fase di bootstrapping di un network IPv6 e permette ad un nodo di effettuare le seguenti operazioni:

- trovare altri nodi nello stesso link.
- determinare l'indirizzo di livello 2 degli altri nodi.
- identificare i router.
- mantenere le informazioni di raggiungibilità verso i nodi con i quali comunica attivamente.

Il protocollo di Neighbor Discovery (di seguito ND) separa i nodi in host e router: i router sono i soli abilitati al forwarding dei paccheti IP ed effettuano operazioni aggiuntive rispetto agli host all'interno del protocollo ND.
Inoltre ND prevede un ulteriore ruolo, l'edge router, specializzato in alcune funzioni complesse del protocollo ND, riducendo così la complessità delle operazioni per i router e gli host. La versione di ND ottimizzata per 6LoWPAN, 6LoWPAN-ND può essere utilizzata anche per distribuire le informazioni di contesto che permettono una migliore efficienza della compressione dell'header context-based.

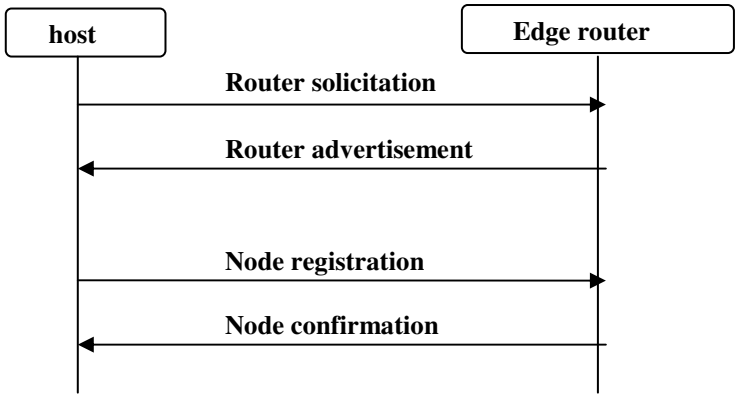


Figura 2.24 6LoWPAN-ND

Il protocollo standard ND prevede l'invio periodico da parte dei router di un pacchetto di router advertisement (RA) che può essere sollecitato con un pacchetto di Router solicitation (RS) da parte del nodo.
La RA non sollecitata dal nodo può contenere informazioni sommarie e se un nodo verifica la presenza di variazioni rispetto alle informazioni possedute, può richiedere un aggiornamento tramite unicast RS alla fonte della RA ed il router risponderà con un RA che include tutto il gruppo di informazioni di prefisso.
Con le informazioni di prefisso inviate tramite RA, il singolo nodo può procedere alla formazione dell'indirizzo, in modalità Stateless Address Autoconfiguration (SAA), utilizzando le informazioni ricevute dalla RA.
Dopo la formazione dell'indirizzo, è necessario verificare la presenza di indirizzi duplicati tramite la Duplicate Address Detection (DAD), che nello standard ND avviene inviando una NS ad un indirizzo multicast definito sulla base dell'indirizzo da validare, mentre in 6LoWPAN-ND avviene utilizzando l'edge router come focal-point per l'operazione di DAD. Ogni edge router gestisce una lista di indirizzi (whiteboard), accessibile ai nodi, che viene alimentata tramite messaggio ICMPv6, di tipo Node Registration (NR) e Node Confirmation (NC): questo processo è detto registration.
Nel Node Confirmation, l'edge-router invia l'elenco degli indirizzi nella propria whiteboard: le entry o bindings nell' elenco devono essere rinfrescate periodicamente, con un messaggio NR dai nodi per rimanere valide.
Nella fase di registrazione è possibile che si verifichino delle collisioni, che in 6LoWPAN-ND possono essere di due tipi:

1. Collisione degli indirizzi: se più nodi cercano di registrare lo stesso indirizzo IPv6 che può andare a buon fine solo per un nodo. Ad ogni registrazione è associata una coppia OII e IPv6 e ad ogni nodo con diverso OII che tenta di registrare un IPv6 già associato, viene negata l'autorizzazione. Questo meccanismo è particolarmente utile per assicurare l'unicità degli indirizzi short a 16-bit.
2. Collisione dell'OII: L'individuazione delle collisioni si basa sull'assunto che gli OII siano globalmente unici, sulla base dell'assegnazione di EUI-64, comunque se un errore porta alla condivisione del medesimo OII da parte di due nodi, il funzionamento di LoWPAN può essere severamente compromesso.

L'edge router archivia le seguenti informazioni di un binding:

| | |
|---|---|
| IPv6 address | Unicast IPv6 address in fase di registrazione da parte del nodo LoWPAN |
| OII | La owner interface identifier del nodo LoWPAN utilizzato nella fase di collision detection |
| Owner Nonce | L'owner nonce fornito durante l'ultima registrazione avvenuta con successo, utilizzato per la duplicate OII detection, è un numero randomico generato ad ogni boot del nodo e serve per distinguere nodi con lo stesso OII. |
| TID | Il transaction identifier è legato all'ultima registrazione andata a buon fine, utilizzato per la duplicate OII detection, è un sequence number che segue un "lollipop scheme", utilizzato insieme all'owner nonce. Ad ogni invio di NR, il nodo incrementa il TID, quando un NC positivo viene ricevuto, se TID è un rookie value (<16) il nodo lo porta a 16 (mature TID): un TID nella parte lineare denota un nodo appena ripartito e non ancora registrato. |
| Primary flag | indica se questo è il primary edge-router, informazione che influenza il ND tra router in situazioni di extended LoWPAN |
| Age/lifetime | Indica quanto tempo è passato dall'ultimo messaggio scambiato. La whiteboard entry viene cancellata quando l'age raggiunge la registration lifetime. |

### 2.12.2 Multihop registration

Il processo di registrazione diventa più complesso quando un nodo non è adiacente ad un edge router. I nodi si possono registrare ad un router confinante se quest'ultimo manifesta la capacità di gestire le registrazioni, tramite il flag M nel suo RA. La registrazione viene gestita dal router rilanciando la NR all'edge-router e la NC di ritorno al nodo.
Al momento del primo messaggio NR, il link-local address è ancora optimistic e per questo il nodo invia il NR con IPv6 non specificato (::). Il router che effettua il relay, prende nota dell'OII nella NR e del source link-layer address e utilizza queste informazioni per inviare successivamente la risposta NC dal edge-router al link-layer del nodo.

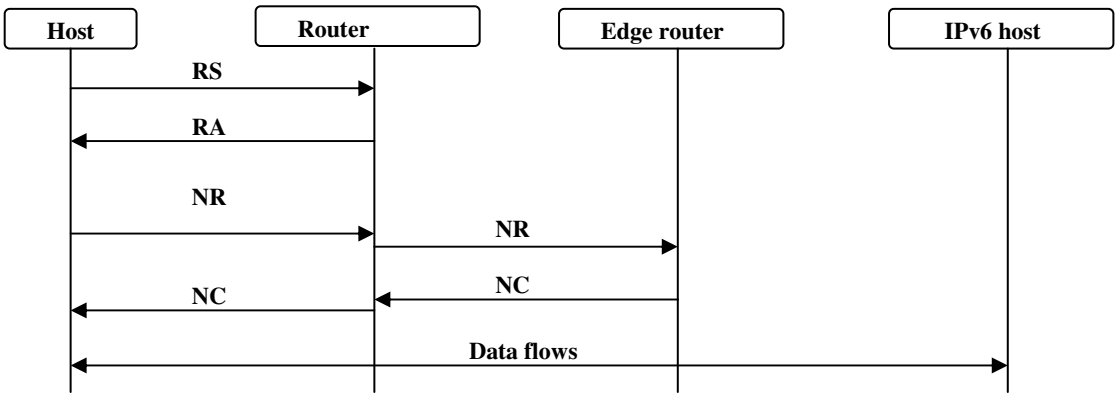


Figura 2.25 Multihop registration

### 2.12.3 Node operation

Il nodo LoWPAN inizia le operazioni autoconfigurando il proprio indirizzo link-local, a partire dell' EUI-64 dell'interfaccia LoWPAN e parte come *optimistic* address, richiedendo conferma con scambio di messaggio NR/NC con l'edge-router, prima di passare allo stato di indirizzo operativo o *preferred*. A questo punto il global prefix è stato acquisito dal nodo, lo stesso scambio di NR/NC può essere utilizzato per registrare l'indirizzo composto dal global-prefix e l''EUI-64 modificato. Al termine il nodo può richiedere un link-layer 16-bit short address all'edge-router, registrando un indirizzo composto da global-prefix e IID composto da PAN ID ed il 16-bit short address assegnato.
Il binding creato nell'edge-router ha un lifetime che costringe il nodo ad inviare periodicamente un NR per rinnovare il binding: se un nodo non riceve un NC, dopo alcuni tentativi e dopo aver cambiato il first-hop router, l'indirizzo ritorna allo stato optimistic ed il processo di registrazione deve ripartire.
Viste le corrispondenze tra il link-layer address ed il local-link address, alcuni nodi fanno uso di questa mappatura per ricavare il link-layer address dall'IPv6 LoWPAN a cui vogliono inviare un pacchetto, perciò un nodo LoWPAN non invia mai un NS per la risoluzione dell'indirizzo, aumentando l'efficienza e riducendo la complessità implementativa.
Il nodo può sapere se raggiungere la destinazione tramite one-hop oppure tramite router utilizzando le informazioni in cache dei nodi confinanti oppure può inviare ottimisticamente il pacchetto all'indirizzo ricavato dall'IPv6 del destinatario.
Il link-layer ACK può essere utilizzato per determinare quando il pacchetto è arrivato a destinazione: in caso di assenza di conferma, il comportamento classico, è l'invio del pacchetto al default router.
LoWPAN router ed edge router utilizzano ICMPv6 destination unreachable per indicare che non è possibile consegnare i pacchetti a destinazione (ICMP Type 1).
Oltre agli indirizzi unicast, i nodi devono supportare l'indirizzo di multicast all-nodes (FF02::1), utilizzato per la ricezione di RA dai router.

### 2.12.4 Router operation

Un router LoWPAN si attiva come ogni altro nodo, inizializza le interface ed i relativi indirizzi e per effettuare il Duplicate Address Detection deve trovare l'edge router oppure un altro router che evidenzia un percorso verso l'edge router.
Dopo l'inizializzazione il router può far partire i servizi di routing e poi inizia a pubblicare i propri servizi agli altri nodi, utilizzando RA ed ascoltando i RS.
I parametri che il router inserisce nei RA sono copie dei parametri ricevuti durante la fase di boot e perciò il router deve continuare a verificare che i parametri che riceve dall'edge router non vengano aggiornati (si incrementa il sequence number in 6LoWPAN prefix number summary option): in tal caso il router aggiornerà i parametri con l'effetto che gli aggiornamenti effettuati dall'edge router vengono diffusi in tutto il LoWPAN.
Un LoWPAN router deve effettuare il relay dei messaggi che i nodi adiacenti inviano all'edge router e rilanciare i messaggio provenienti dall'edge router verso il nodo stesso.

Nel messaggio di NR ricevuto da un nodo adiacente, il router modifica il campo Code ad 1, per indicare che il messaggio è rilanciato, configura l'IPv6 source address con il proprio, ricalcola il checksum, pone Hop Limit a 255 e di solito invia il messaggio modificato all'anycast address 6LOWPAN_ER.
I nodi che in fase di boot inviano il messaggio iniziale di NR, utilizzano come source IPv6 l'indirizzo unspecified :: e per permettere ad un router di consegnare i messaggio di NC al nodo, il router tiene traccia dello stato del nodo, registrando la mappatura dell' OII con l'indirizzo di link-layer: quando il NC giunge al router, viene riportato a 0, l'hop limit viene impostato a 255 ed utilizza l'informazione di stato per ricavare il link-local IPv6 desination address ed il link-layer per restituire il NC al nodo. In questo momento il router alimenta la propria cache con le informazioni del nodo confinante.

### 2.12.5 Edge router operation

L'edge router, a differenza dagli altri elementi della rete LoWPAN, è connesso anche alla rete IPv6 tramite un'altra interfaccia: la connessione può essere di tipo backhaul link e in questo caso la rete è di tipo *simple LoWPAN* oppure può essere un backbone link, che connette anche un altro edge router e in questo caso la rete è di tipo *Extended LoWPAN*.
Il caso di *Extended LoWPAN* richiede coordinamento tra gli edge router.
L'edge router è un router con le seguenti funzionalità aggiuntive, che richiedono un'attività di configurazione addizionale:

1. L'edge router è la fonte dei parametri che vengono inviati nel LoWPAN tramite RA, incluso il LoWPAN prefix e le altre informazioni di context che servono per l'header compression context-based.
2. L'edge router gestisce la whiteboard e gli algoritmi di detection.
3. L'edge router effettua routing tra la rete IPv6 e LoWPAN ed implementa funzionalità di protezione nella fase di routing: filtra messaggi destinati a nodi non conosciuti e può inoltre implementare un firewall o altre forme di packet filtering.

Nel caso di *Extended LoWPAN,* il cui supporto è opzionale, più router sono interconnessi con un backbone link che supporta il multicast, come ad esempio Ethernet.
Il backbone link ha lo stesso prefix del LoWPAN e per questo l'edge router deve effettuare la traduzione tra i messaggi 6LowPAN-ND nell'interfaccia LoWPAN e gli standard ND nel backbone link.
Gli algoritmi di detection sono estesi al backbone link, mediante l'utilizzo di NS e NA del protocollo ND standard, infatti l'edge router rappresenta tutti i nodi che si sono registrati, non solo se stesso.
Il NS nel backbone link è gestito direttamente dall'edge router e non è propagato nel LoWPAN e per questo l'edge router deve associarsi al solicited-node multicast address nel backbone link per tutti gli indirizzi dei nodi registrati e deve rispondere ad ogni NS con neighbor advertisement che indica nel target link-layer address option l'indirizzo link-layer sul backbone link dell'edge router.
I nodi si possono muovere all'interno del LoWPAN ed inviando aggiornamenti del NR all'indirizzo 6LOWPAN_ER anycast, possono raggiungere edge router diversi da

quello della registrazione iniziale: il nuovo edge router semplicemente prende in carico la nuova registrazione, ignaro della migrazione ed il vecchio edge router lascia la registrazione al router al quale è giunta la nuova registrazione.
Tra gli edge router, sul backbone link, vengono utilizzati i messaggi ND standard come Neighbor Solicitation (NS) e Neighbor Advertisement (NA): per permettere l'utilizzo di owner interface identifier (OII) e transaction Id (TID) negli algoritmi di conflict detection anche sul backbone link, si estendono NS/NA con l'opzione di *owner interface identifier*.

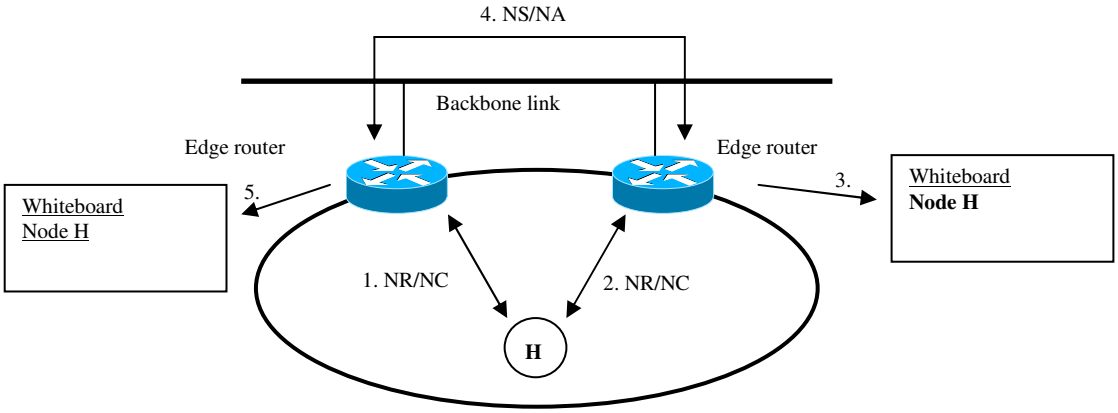


Figura 2.26 Edge-router operations

## 2.13 Security

Le reti wireless per natura sono particolarmente esposte ad hacker, che possono provare ad ascoltare ed inviare pacchetti in rete e non c'è cosa peggiore di notizie sulla insicurezza di una tecnologia emergente per comprometterne il futuro.
Gli obiettivi da raggiungere sono tre:

1. confidentiality: i dati non devono essere disponibili a listener non autorizzati, perciò spesso si ricorre all'encryption.
2. integrity: i dati non possono essere alterati da utenti non autorizzati, perciò spesso si aggiunge al messaggio un cryptographic integrity check. Un messaggio inviato da un utente non autorizzato può essere equivalente ad un messaggio inviato da un utente autorizzato, che viene poi modificato, si deduce perciò che l'autenticazione è strettamente correlata con l'integrity.
3. availability: il sistema non deve essere oggetto di attacchi di tipo denial of service. Un sistema wireless può essere sempre oggetto di jamming a livello radio, ma tale evento può essere sempre localizzato facilmente ed interrotto. Più complicata è la gestione di denial of service da source che non possono essere facilmente individuabili, come ad esempio il "ping of death" che può essere inviato per inficiare il funzionamento di sistemi operativi non protetti.

Gli obiettivi di sicurezza possono essere ricavati direttamente dai requirement applicativi e la loro definizione permette di analizzare il thread model, cioè cosa può fare un hacker per indebolire gli obiettivi di sicurezza.

Il threat model di un sistema wireless come 6LoWPAN è differente dal threat model generico dei protocolli Internet: si suppone che l'hacker possa avere il pieno controllo del canale trasmissivo, possa accedere ed inviare i pacchetti.
Senza il supporto delle crittografia non c'è modo di proteggere le comunicazioni 6LoWPAN da hacking, inoltre la natura dei sensori distribuiti in zone limitate, permette di ottenere facilmente il controllo di un nodo e visti i costi limitati, è difficile proteggere i nodi da hacking fisico.

### 2.13.1 Protezione a livello 2

La possibilità di utilizzare meccanismi di protezione del layer 2, in aggiunta alla security delle soluzioni end-to-end, fornisce un livello di ottimizzazione che può garantire un trasporto affidabile e maggiormente protetto da attacchi alla confidentiality ed integrity.
Sulla base dei problemi di security del IEEE 802.11, IEEE802.15.4 ha stabilito il supporto da parte di ogni nodo di meccanismi avanzati di crittografia, supportati dalla maggior parte dei chip IEEE802.15.4.
L'algoritmo di encryption scelto è AES con CBC-MAC (CCM), fornendo oltre all'encryption anche una soluzione di integrity check.
Entrando nello specifico, AES/CCM cripta messaggi di dimensione m con eventuali dati autenticati di dimensione a, utilizzando un chiave K e un nonce N. Il parametro L controlla il numero di byte utilizzato per estrarre i blocchi di byte dal messaggio e m deve essere inferiore a $2^{8L}$: nel caso di IEEE 802.15.4 è utilizzato il valore L=2.
Il nonce N è di lunghezza 15-L, 13 byte nel caso IEEE 802.15.4, non è secretato ma deve essere utilizzato una volta sola, infatti se si ha accesso a due messaggi con il medesimo valore di K e N la sicurezza di AES/CCM è perduta.
Il risultato di AES/CCM è un messaggio cryptato della stessa lunghezza m esattamente come il valore di autenticazione di M byte, dove M è un parametro che può essere un qualsiasi numero pari tra 4 e 16, limitato a 0 (nessuna autenticazione), 4, 8 o 16 byte in IEEE 802.15.4. Il valore di autenticazione può essere creato correttamente se K è conosciuto e può essere verificato dal ricevente, per assicurarsi che m ed a non siano stati alterati.
IEEE 802.15.4 ricava i 13 byte di nonce dagli 8 byte del full address del dispositivo che origina la frame cryptata (pacchetto L2), 4 byte frame counter ed un byte occupato da IEEE 802.15.4 security level.
Il source address ed il security level si ripetono, perciò la sicurezza dell'intero sistema si appoggia ai restanti 4 byte del frame counter, che permette l'invio di 232 frame cryptate prima che la chiave venga riutilizzata, circa $2^{22}$ secondi o 7 settimane, ipotizzando che il nodo monopolizzi il canale e lo utilizzi alla massima banda data da IEEE 802.15.4 con frame di 32 byte.

### 2.13.2 Protezione a livello 3

Quando i dati lasciano LoWPAN perdono della protezione a livello link-layer e diventano vulnerabili in ogni punto che deve gestirli.
Solo mediante una security end-to-end si può proteggere la conversazione tra i due nodi ed è elemento fondamentale del protocollo IPv6, trasportato in IPv4 con il nome di IPSec.

I nodi LoWPAN hanno un hardware progettato per AES/CCM encryption, decryption e calcolazione dell'integrity check ed è disponibile anche per i layer superiori al link-layer: AES/CCM è perciò il candidato per ESP, nella protezione end-to-end.
Encrypted payload e ICV con AES/CCM è composto da un initialization vector (IV) di 8 byte, che insieme a 3 byte di salt della security association, producono 11 byte nonce che vengono elaborati da CCM (AES/CCM per ESP utilizza CCM con L=4 per fornire supporto alle large jumbogrums), poi l'encrypted payload e poi ICV, che può avere una lunghezza di 8, 12 o 16 byte, da definire nella security association.
L'overhead per pacchetto può essere di 1-4 byte per il paddinge 8 byte per l'initialization vector.

## 2.14 Mobilità

Le componenti di un sistema 6LoWPAN possono spostarsi nello spazio a causa del movimento da parte dei sensori legati ad oggetti mobili, ad esempio in ambiente industriale, in tal caso si parla di node mobility.

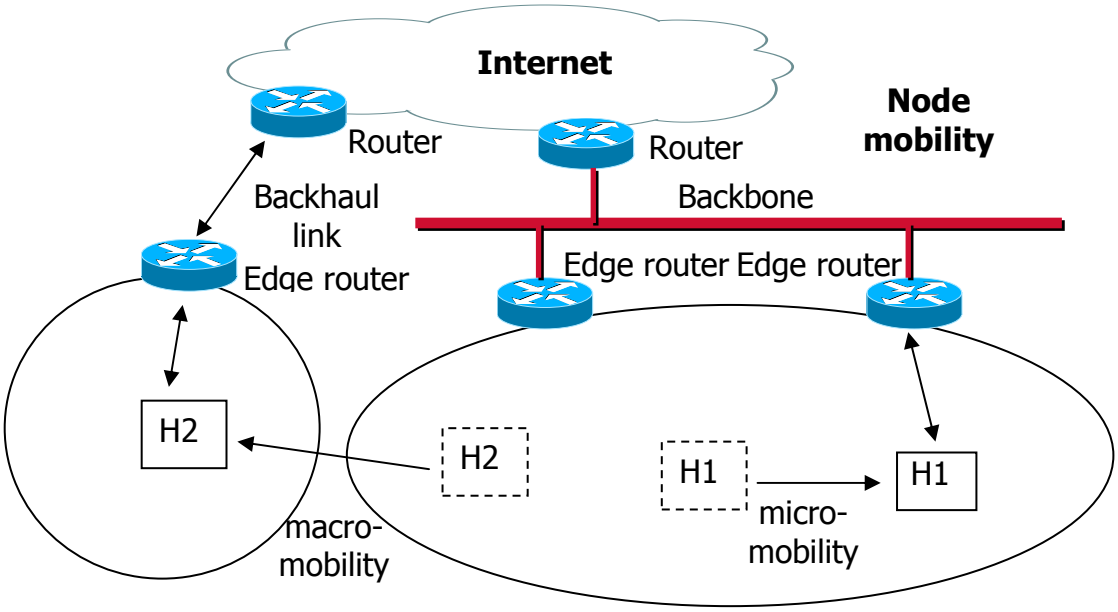


Figura 2.27 Mobility

La mobilità in una rete 6LoWPAN può essere causata anche dall' edge router, che varia il punto di connessione ad Internet, in questo caso si parla di network mobility.

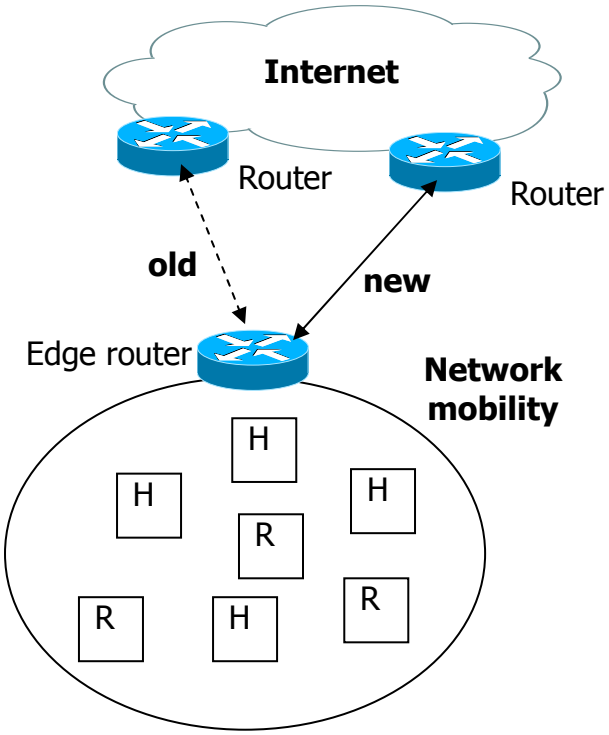


Figura 2.28 Network mobility

In altri casi si può sperimentare una variazione della topologia, anche senza movimenti da parte delle componenti del sistema 6LoWPAN: un esempio è dato dall'interferenza ambientale sui canali radio, che può influenzare la connettività tra i nodi e può forzare i nodi a ricercare percorsi alternativi. Inoltre i sensori che esauriscono l'energia a disposizione o che effettuano un lungo sleep cycle possono influenzare la topologia della rete.
Si può distinguere tra il processo di Roaming, quando il nodo si sposta da un network all'altro in assenza di flussi di pacchetti ed il processo di handover, quando il nodo cambia punto di connessione in presenza di flussi di pacchetti.
In presenza di node mobility è importante poter gestire i casi di mobilità tra diversi edge router all'interno dello stesso LoWPAN (micro-mobility) oppure tra LoWPAN distinti con modifica dell'estensione dell'indirizzo (macro-mobility).
Genericamente quando un nodo varia il proprio punto di aggancio alla rete 6LoWPAN deve eseguire le seguenti operazioni:

1. stabilire un link per effettuare la fase di commissioning.
2. configurare un indirizzo IPv6, tramite la fase di bootstrapping.
3. configurare i paramentri di security e firewalling.
4. verificare se le entry del DNS sono appropriate oppure se devono essere aggiornate.
5. notificare la variazione al layer applicativo ed aggiornare ogni identificatore a livello applicativo ed ogni registrazione.

Le tecnologie  a basso consumo di energia come IEEE 802.15.4 tendono a demandare al network layer la gestione della mobilità ed in 6LoWPAN attualmente non sono definite tecniche per la micro-mobility nel caso link-layer mesh-under.
Neighbor Discovery per 6LoWPAN prevede alcune funzioni per la gestione della micro-mobility nel caso di Extended LoWPAN, tramite una tecnica di ND proxy

combinata con la sincronizzazione della whiteboard tra gli edge router che permette al nodo di mantenere lo stesso indirizzo IPv6.
Per quanto riguarda le problematiche legate alla macro mobility, il modo più semplice è la ripartenza del nodo ma questo ha pesanti implicazioni dal punto di vista applicativo, specialmente se il nodo agisce come server.
E' da notare che molte applicazioni in reti 6LoWPAN utilizzano UDP come trasporto, protocollo che si può adeguare meglio alla modifica dell'indirizzo IPv6 ma per mantenere il livello applicativo è necessario l'utilizzo di un identificatore del nodo che non sia l'indirizzo IPv6, ad esempio l'EUI-64, un URI (universal unique identifier) o un domain name che viene risolto tramite DNS.
La soluzione MIPv6, non risulta adeguata a causa delle ridotte risorse in termini energetici e computazionali dei sensori 6LoWPAN, mentre risultano interessanti le soluzioni che demandano all'edge router la gestione della mobilità del singolo nodo e del network (PMIPv6, NEMO).

## 2.15 Routing

IP routing ha come scopo la gestione e l'aggiornamento di tabelle che indicano ad ogni nodo il next-hop per l'instradamento dei pacchetti, sulla base dell' IP di destinazione.
IP routing utilizza molteplici tecniche per l'aggiornamento delle tabelle ed in questo paragrafo approfondiremo le tematiche legate al routing in ambito 6LoWPAN, alternativa alle tecniche di instradamento a livello link-layer, come link-layer mesh e LoWPAN Mesh-under già discusse in precedenza.
IP routing può utilizzare le seguenti caratteristiche peculiari di 6LowPAN:

1. i router 6LoWPAN eseguono il forwarding del pacchetto sulla stessa interfaccia dalla quale lo ricevono e questo meccanismo è utilizzato per mettere in connessione due nodi che non riescono a dialogare tramite un solo hop.
2. in LoWPAN l'indirizzamento è flat e tutti i nodi condividono lo stesso IPv6 prefix, perciò le tabelle di routing contengono solo le informazioni per il raggiungimento dei nodi in LoWPAN e le default route.
3. la rete LoWPAN è una stub network e perciò non è una rete di transito.

Ne consegue che in 6LoWPAN possono essere presenti due tipologie di routing: intra-LoWPAN routing, gestito dai nodi con ruolo di router e border routing, gestito dall'edge router al confine tra la rete 6LoWPAN ed IPv6.
Inoltre, in rete 6LoWPAN il protocollo Neighbor Discovery può contribuire al processo di forwarding nel caso di Extended LoWPAN, essendo coinvolto nelle decisioni di next-hop per i nodi di tipo host, nella gestione della informazioni sui nodi confinanti, destination cache e router cache e giocando un ruolo fondamentale nel Neighbor Unreachability Detection.

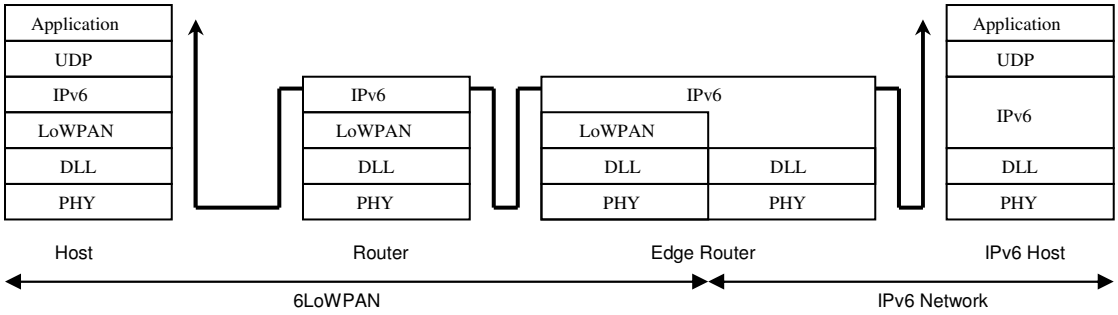


Figura 2.29 Routing

Il protocollo di routing deve rispettare alcuni requirements, che sono in conflitto tra loro:

1. ottimizzazione nell'utilizzo di energia nei nodi router.
2. supporto delle fasi di sleep mode del nodo.
3. supporto della quality of service.
4. supporto delle diverse tipologie di indirizzamento.
5. integrazione con la mobilità.
6. ottimizzazione del consumo di memoria e banda.

Di base sono state definite 4 aree di applicazione: urban networks, industrial networks, building networks e home networks ed in base all'area di applicazione dei sensori, alcuni requirements del protocollo di routing verranno enfatizzati, mentre altri saranno meno rilevanti.

### 2.15.1 Caratteristiche dei protocolli di routing

I protocolli di routing 6LoWPAN possono essere distinti in distance-vector routing e link-state routing, mentre le informazioni di routing possono essere aggiornate a priori (proactive) oppure on-demand (reactive).
Gli algoritmi di distance-vector routing si basano su varianti dell'algoritmo di Bellman-Ford e di base ad ogni link viene assegnato un costo, utilizzando metriche di routing e nell'instradamento dei pacchetti viene utilizzato il percorso con minor costo per il raggiungimento dell'indirizzo di destinazione.
Gli algoritmi di tipo link-state richiedono che ogni nodo acquisisca la conoscenza completa dell'intero network e per questo ogni nodo deve inviare in rete le informazioni locali che conosce.
In base alle informazioni ricevute, un nodo è in grado di costruire la propria visione della rete con le calcolazioni dei percorsi più veloci verso gli altri nodi: questa architettura viene aggiornata continuamente, richiedendo un'elevata quantità di risorse del nodo e della rete.
Se le informazioni di routing a livello di singolo nodo vengono costruite prima che servano al processo di routing, è necessario un processo di acquisizione proattivo dalla rete.
Alcuni esempi di proactive routing protocol sono dati da OLSR, optimized link-state routing e TBRPF, topology dissemination based on reverse-path forwarding.
Il pregio dell'approccio proattivo è la velocità del processo di forwarding del pacchetto che si basa su informazioni di routing già acquisite ed elaborate, mentre il

costo deriva dall'intensa attività di segnalazione, specialmente nel caso di topoligie con frequenti cambiamenti.
Nel caso le informazioni di routing vengano acquisite in modalità reattiva, cioè quando servono, è necessario l'utilizzo di un processo di route discovery, nel momento in cui il nodo riceve un pacchetto con destinazione non conosciuta dal processo di routing.
Alcuni esempi di reactive routing protocol sono dati da AODV, ad hoc on-demand distance vector e DYMO, dynamic MANET on-demand e ZigBee routing protocol, derivato da AODV.
Il pregio dell'approccio reattivo è dato dalla crescita del traffico di segnalazione in rete solo nel momento di necessità delle informazioni specifiche, con conseguente lentezza del processo di forwarding, particolarmente adatto alle reti con frequenti cambiamenti di topologia.
Nella fase di selezione del miglior percorso nel processo di routing vengono utilizzate le metriche di routing che comunemente includono numero di hop, larghezza di banda, MTU, ritardo, affidabilità, che adattate alle reti 6LoWPAN diventano le seguenti:

| Metrica | Tipologia della metrica | Descrizione |
|---|---|---|
| Node memory | Quantitativa, statica | Memoria disponibile per le informazioni di routing nel nodo |
| Node CPU | Quantitativa, statica | Capacità computazionale del nodo |
| Node energy | Quantitativa, dinamica | Energia residua per nodi alimentati da batteria |
| Node overload | Quantitativa, dinamica | Indicazione del carico di rete del nodo |
| Link throughput | Quantitativa, dinamica | Throughput totale e disponibile del link |
| Link latency | Quantitativa, dinamica | Latenza attuale e range per il link |
| Link reliability | Quantitativa, dinamica | Average packet error rate |
| Link coloring | Quantitativa, statica | Informazione per la selezione del link per tipologie di traffico |

Queste metriche vengono utilizzate sia per la costruzione della topologia di rete che per il forwarding dei pacchetti.

### 2.15.2 I protocolli di routing MANET

Il gruppo di lavoro IETF MANET, mobile ad hoc network era stato costituito per studiare i requirement e definire le soluzioni per le applicazioni in ambiente wireless ad hoc ed ha prodotto alcuni protocolli di routing: AODV, DYMO e OLSR.
Il protocollo AODV, ad hoc on-demand distance vector è un protocollo di tipo distance-vector reactive, mantiene solo gli instradamenti utilizzati e prevede metodologie per assicurare operatività loop-free.

Il discovery degli instradamenti utilizzando piccoli messaggi, route request (RREQ) che vengono inviati in broadcast nella rete per trovare il corretto instradamento verso la destinazione richiesta.
Un router intermedio o la destinazione stessa rispondono con un messaggio di route reply (RREP). Il messaggio di route error (RERR) è utilizzato per notificare link non più disponibili lungo il percorso.
Il protocollo DYMO, dynamic MANET on-demand utilizza lo stesso meccanismo di discovery di AODV e ne migliora la convergenza in topologie dinamiche.
Dymo utilizza il formato di pacchetti MANET e registra la funzionalità dei nodi host e router nella fase di discovery.
Il protocollo OLSR, optimized link-state routing è di tipologia proactvie link-state ed applica una ottimizzazione del classico algoritmo link-state in uso nelle reti mobili ad hoc.
In OLSR il flusso delle informazioni è controllato dall'utilizzo di nodi selezionati per il multipoint relay, che vengono utilizzati come router intermedi. OLSR utilizza il formato MANET e le tecniche two-hop ND.
OLSR è adatto a reti ad hoc relativamente statiche e non è ottimizzato per 6LoWPAN a causa dell'elevato traffico di segnalazione e di routing state.

### 2.15.3 I protocolli di routing ROLL

Il gruppo di lavoro ROLL in IETF è stato costituito per studiare i requirement e standardizzare un routing protocol che soddisfi le necessità di routing in reti low power and lossy (LLN), tra le quali compare IEEE 802.15.4, Bluetooth, low-power WiFi e PLC.
Le reti considerate dal gruppo di lavoro ROLL hanno queste caratteristiche peculiari:

1. il traffico non è solo peer-to-peer unicast, ma spesso è peer-to-multipoint o multipoit-to-point
2. router in LLN hanno memoria limitata
3. LLN devono essere ottimizzati per il consumo energetico
4. LLN implementati su link con dimensioni delle frame limitate
5. sicurezza e gestibilità sono problematiche rilevanti in LLN
6. spazio di applicazione eterogeneo

Di seguito elenchiamo le attività di analisi in essere nel ROLL Working group:

- Metriche di routing: sono state definite ma è necessario definire a livello di algoritmo quale metrica in base allo specifico ambito applicativo.
- Architettura: i requirement architetturali di base per ROLL sono in fase di definizione, ma prevedono che LLN sia una stub network connessa ad Internet con supporto per la connessione multi-point, con trafico preminente tra LLN e l'esterno, mentre il traffico tra i nodi è meno fraquente.
- Security: un security framework ed alcune considerazioni per la gestione sono in fase di definizione.
- Protocollo: l'obiettivo è la definizione di un routing protocol che possa essere applicato nelle 4 aree applicative definite da ROLL e la fase iniziale è conclusa.

Il protocollo ROLL può essere catalogato come algoritmo proactive distance-vector con alcune funzionalità avanzate, agisce nell'ambito dell' LLN e termina sull'LLN border router (LBR), che spesso coincide con il punto dove è implementata la funzione LoWPAN edge router.
Il protocol routing utilizza una struttura a grafo tra i nodi e l'LBR che deve essere costruita inizialmente con una ridotta segnalazione ed in fase succesiva per ogni nodo viene mantenuto un percorso di upstream verso LBR e downstream dall'LBR.
ROLL definisce una metrica di routing granulare chiamata depth, che viene utilizzata per la definizione del grafo e per evitare loop.
Inoltre ROLL definisce due periodi diversi: la fase inziale di route-setup, nella quale vengono utilizzate metriche statiche o lentamente variabili e la fase di operatività di packet-forwarding, nella quale si utilizzano metriche dinamiche.
La fase di costruzione della topologia sfrutta l'invio dei messaggi di ND utilizzandoli come messaggi di segnalazione, evitando l'utilizzo di pacchetti aggiuntivi di segnalazione. Le informazioni relative al grafo vengono inviate dal LBR nel network ed i router si associano al grafo definendo i default next-hop router verso LBR, tramite regole specifiche, che permettono il flusso in upstream. La topologia è gestita tramite messaggi periodici inviati da LBR ai nodi e propagati da ogni router. Le informazioni in merito al downstream derivano dall'obbligo da parte dei nodi di disseminare le informazioni di upstream path cost verso LBR, arricchendo una topologia a grafo con informazioni aggiuntive.

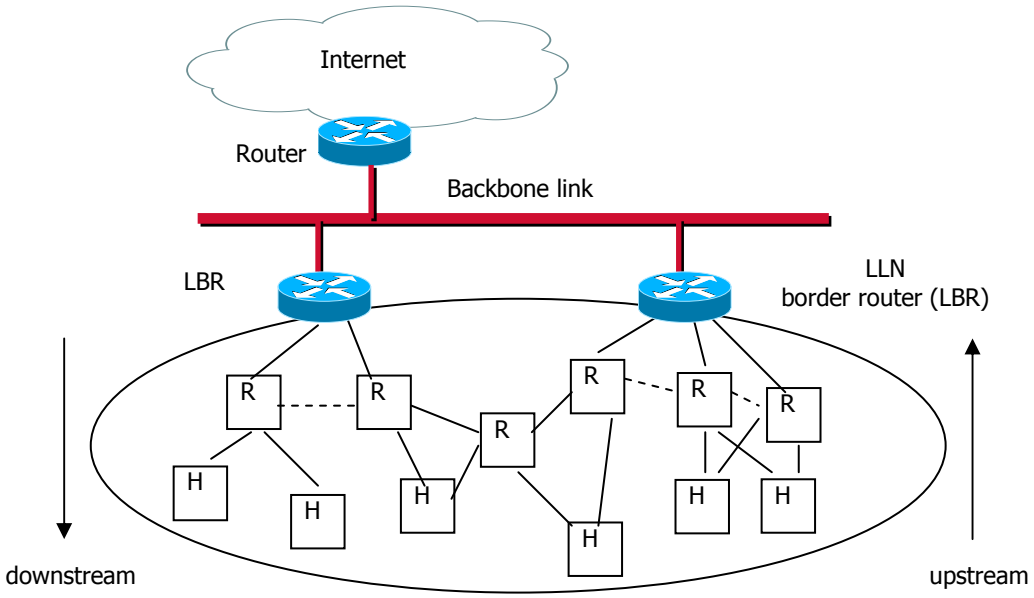


Figura 2.30 Roll

Le informazioni possono essere disseminate utilizzando i messaggi ND e le informazioni servono ad alimentare le informazioni utili per l'algoritmo di distance-vector nei router intermedi e per LBR per ottenere informazioni globali in merito all'LLN.
La topologia di base costruita da ROLL permette il forwarding dei pacchetti tra i nodi LLN e LBR (upstream) mentre il meccanismo di distribuzione delle informazioni verso LBR permette il forwarding dei pacchetti da LBR verso i nodi LLN.

La ridondanza della struttura è assicurata dalla presenza di più di una default route, selezionata al momento dell'invio dell'informazione sulla base delle metriche a disposizione del nodo.
L'invio di traffico da parte di un nodo, viene effettuato al migliore default router e se l'operazione non va a buon fine, il nodo utilizza il default router che le metriche indicano come successivo. Il forwarding downstream può essere effettuato tramite hop-by-hop distance-vector o source routing o una combinazione dei due.
Per raggiungere alcuni requisiti identificati da ROLL sono dispnibili alcune tecniche di ottimizzazione tra le quali traffic engineering, node-to-node flows e mobility support.

### 2.15.4 Border routing

I protocolli Roll sono stati ottimizzati per le sessioni tra Internet e LoWPAN, tramite LLN border router, tipicamente edge router nel caso 6LoWPAN.
L'edge router dovrà integrare diversi protocolli di routing presenti nelle due interfacce e possiamo distinguere alcuni casi:

1. Simple LoWPAN: in questo caso c'è un solo edge router e la subnet dell'interfaccia LoWPAN è diversa da quella dell'interfaccia IPv6, viene utilizzata il prefix-based routing e tranne in casi particolari non è necessario un protocollo di routing nell'interfaccia IPv6.
2. Extended LoWPAN: l'interfaccia LoWPAN ed IPv6 sono nella stessa subnet perciò il routing può esssere effettuato tramite le entry relative alla destination. L'utilizzo delle whiteboard nell'edge router rappresenta il modo più semplice di effettuare il routing.

Il border routing potrebbe richiedere redistribuzione tra protocolli di routing diversi e dato che 6LoWPAN è uno stub network, la redistribuzione avverrà sempre dall'interfaccia LoWPAN alla rete IP. I candidati per il border routing in combinazione con gli algoritmi ROLL, includono OLSR e OSPF.

# 3. Il sistema operativo Contiki

## 3.1 Introduzione

Le reti di sensori wireless sono composte da un elevato numero di dispositivi che comunicano tra loro in modo non strutturato, creando autonomamente un network wireless, veicolo della comunicazione.
Le risorse a disposizione dei sensori solitamente sono limitate, l'alimentazione fornita da batterie o celle solari è ridotta e limitata nel tempo ed il requisito di limitatazione dei costi del sonsore complica ulteriormente l'architettura.
Contiki [12], il cui nome deriva dalla zattera di Kon-Tiki che attraversò l'oceano Pacifico con risorse minime, è un sistema operativo implementato in linguaggio C, event-driven, con preemptive multi-threading opzionale che può essere applicato a singoli processi.
Contiki è disponibile per molteplici piattaforme (Sky/TelosB, Atmel AVR, etc.) che hanno in comune l'architettura della CPU basato su un modello di memoria senza segmentazione e senza meccanismi di protezione. Il codice è installato nella ROM riprogrammabile (EEPROM) ed in RAM. Le uniche astrazioni fornite dal codice di base sono il multiplexing della CPU ed il supporto per il caricamento dinamico di applicazioni e servizi. Per il resto le altre astrazioni sono implementate tramite librerie, come ad esempio il preemptive multi-threading.

## 3.2 Caricamento del codice in run-time

Nelle reti di sensori wireless, composte da un elevato numero di dispositivi, la possibilità di caricare codice tramite il network risulta rilevante, specialmente nei casi in cui è difficile recuperare fisicamente i sensori, per aggiornare le applicazioni e per risolvere velocemente i bug del codice.
Sono disponibili alcuni metodi per il caricamento dinamico del codice, ma comune a tutti vi è la necessità di limitare il numero di bytes che transitano in rete, per ridurre la quantità di energia necessaria.
Molti sistemi operativi richiedono il caricamento completo dell'immagine binaria, che include il sistema operativo, le librerie di sistema e le applicazioni che funzionano sopra al sistema operativo.
Contiki permette il caricamento e la sostituzione dinamica di programmi e servizi [6], in modo flessibile, mantenendo il sistema base leggero e compatto.
Molto spesso le singole applicazioni sono di dimensione trascurabile rispetto all'intero sistema e perciò richiedono meno energia nella trasmissione nel network ed il tempo di trasferimento risulta sensibilmente inferiore.
I programmi caricabili sono implementati tramite una funzione di rilocazione run-time ed il codice contiene le informazioni necessarie per la rilocazione: quando un programma è caricato nel sistema, il loader tenta di rendere disponibili le risorse secondo le informazioni nel codice e se le risorse non sono disponibili l'esecuzione viene arrestata. Dopo che il codice viene caricato, viene chiamata una funzione di inizializzazione.

## 3.3 Event-driven o multi-threaded

In dispositivi con memoria limitata, un modello con operazioni multi-threaded può consumare un' elevata parte di memoria, infatti ogni thread deve avere il proprio stack, dato che è difficile conoscere in anticipo la quantità richiesta e perciò deve essere sovradimensionato e caricato quando il thread è eseguito. La memoria contenuta nello stack non può essere condivisa con altri thread.
Per fornire supporto per processi concorrenti, senza la necessità di meccanismi di protezione e stack per thread, Contiki implementa un approccio event-driven: in sintesi i processi sono chiamati a gestire gli eventi e raggiungere il completamento. Il gestore dell'evento non può essere bloccato e perciò tutti i processi possono condividere lo stesso stack, implementato tramite una memoria limitata.
Questo modello non è privo di problemi, infatti, non tutti i programmi possono essere impostati come state-machine.
Inoltre un task che richiede un elevato numero di operazioni, come le operazioni di crittografia, monopolizza la CPU, rendendo il sistema incapace di gestire nuovi eventi, problema che non si presenta nei sistemi multi-thread.
Per risolvere questo problema, Contiki ha adottato un modello ibrido: il sistema è event-driven ma il preemptive multi-threading è implementabile con una libreria, che può essere linkata nel programma che la richiede esplicitamente.

## 3.4 Overview del sistema

Contiki è composto delle seguenti componenti: il kernel, le librerie, il program-loader ed un insieme di processi.
Un processo può essere un'applicazione o un servizio ed un servizio implementa funzionalità che possono essere utilizzate da più applicazioni. Tutti i processi possono essere dinamicamente caricati nel run-time.
La comunicazione tra i processi avviene tramite il kernel che non fornisce un livello di astrazione per l'hardware, ma permette ai drivers ed alle applicazioni di comunicare direttamente con l'hardware.
Un processo è definito da un gestore degli eventi o da una funzione opzionale di "poll handler". Lo stato del processo è mantenuta in una memoria privata ed il kernel mantiene solo un puntatore allo stato del processo.
La comunicazione tra i processi avviene tramite il posting di eventi.

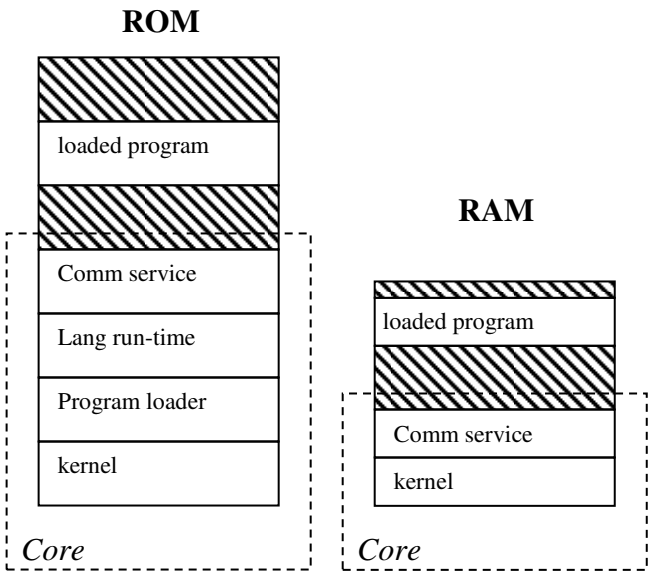


Figura 3.1 Contiki

Contiki è partizionato in due parti: il core ed i loaded programs. Il partizionamento è realizzato in fase di compilazione e tipicamente il core è composto dal kernel, il program loader, le più comuni parti del language run-time, le librerie di supporto e lo stack di comunicazione con i driver per l'hardware di comunicazione.
Il core è compilato in una singola immagine binaria che non viene generalmente modificata dopo il deployment, anche se con un boot loader speciale è possibile sovrascrivere il core oppure applicare patch.
Il program loader può caricare i programmi nel sistema sia tramite lo stack di comunicazione oppure utilizzando l'EEPROM locale: tipicamente un programma viene prima posizionato nella EEPROM e poi caricato nella code memory.

## 3.5 L'architettura del kernel

Il kernel del Contiki consiste di uno schedulatore di eventi che assegna gli eventi ai processi e periodicamente interpella i polling handlers dei processi.
L'esecuzione dei processi è scatenata dagli eventi distribuiti dal kernel oppure tramite il meccanismo del polling ed il kernel non sospende (preemption) un event handler scheludato, che deve perciò giungere a completamento, anche se l'event handler contiene meccanismi che permettono la sospensione (preemption).
Gli eventi supportati dal kernel sono di due tipi: asincroni e sincroni.
Gli eventi asincroni vengono messi in coda dal kernel e distribuiti con leggero ritardo, mentre gli eventi sincroni causano l'immediata schedulazione del processo incaricato della gestione.
In aggiunta alla gestione dell'evento il kernel fornisce il meccanismo del polling che può essere visto come un evento ad elevata priorità che viene gestito in mezzo a due eventi asincroni. Il polling è utilizzato dai processi che necessitano di verificare lo stato dell'hardware e quando un poll è schedualto, tutti i processi che implementano il poll handler vengono eseguiti, in base alla loro priorità.
Il Contiki utilizza un singolo stack condiviso tra tutti i processi e l'utilizzo di eventi asincroni riduce lo spazio utilizzato, dato che lo spazio è riutilizzato ad ogni gestione di nuovo evento.

La schedulazione degli eventi è effettuata in un unico livello e nessun evento può sospendere altri eventi.
L'unica possibilità di sospensione di un evento è data da interrupt che normalmente sono implementati tramite hardware interrupt oppure tramite esecuzioni real-time.
Contiki non permette che i gestori di interrupt possano generare eventi, per evitare "race-conditions", mentre è possibile richiedere un poll event tramite un flag.
Dal punto di vista della gestione delle risorse, il kernel non contiene alcuna funzione di risparmio dell'energia ma ne lascia l'implementazione alle applicazioni, che possono accedere ad informazioni rese disponibili dal kernel, come ad esempio la dimensione della coda degli eventi, che permette all'applicazione di porre in shut il processore, se non necessario. In caso di interrupt, il processore riprende le funzioni ed un poll handler gestirà l'evento esterno.

## 3.6 Services

Un service è un processo che implementa una funzionalità che può essere utilizzata da altri processi, come una libreria condivisa. Un service può essere dinamicamente sostituito durante il run-time e perciò deve essere linkato dinamicamente.
I servizi sono gestiti da un service layer che tiene traccia dei servizi in esecuzione.
Un servizio è identificato da una stringa che lo descrive ed è costituito da un service interface e da un processo che implementa l'interface. Il service interface consiste di un numero di versione ed una tabella con i puntatori alle funzioni che implementano l'interfaccia.
Le applicazioni utilizzano una service stub library per comunicare con il servizio, la quale utilizza il service layer per trovare il servizio ed ottenerne il process ID per le successive richieste. In particolare, la prima volta che il servizio è richiamato, la service interface stub lo ricerca nella service layer e se il servizio esiste viene ritornato un puntatore alla service interface, che contiene un numero di versione e puntatori a tutte le implementazioni delle funzioni contenute nel service. Se i numeri di versione coincidono, l'interface stub richiama l'implementazione delle funzioni richieste.

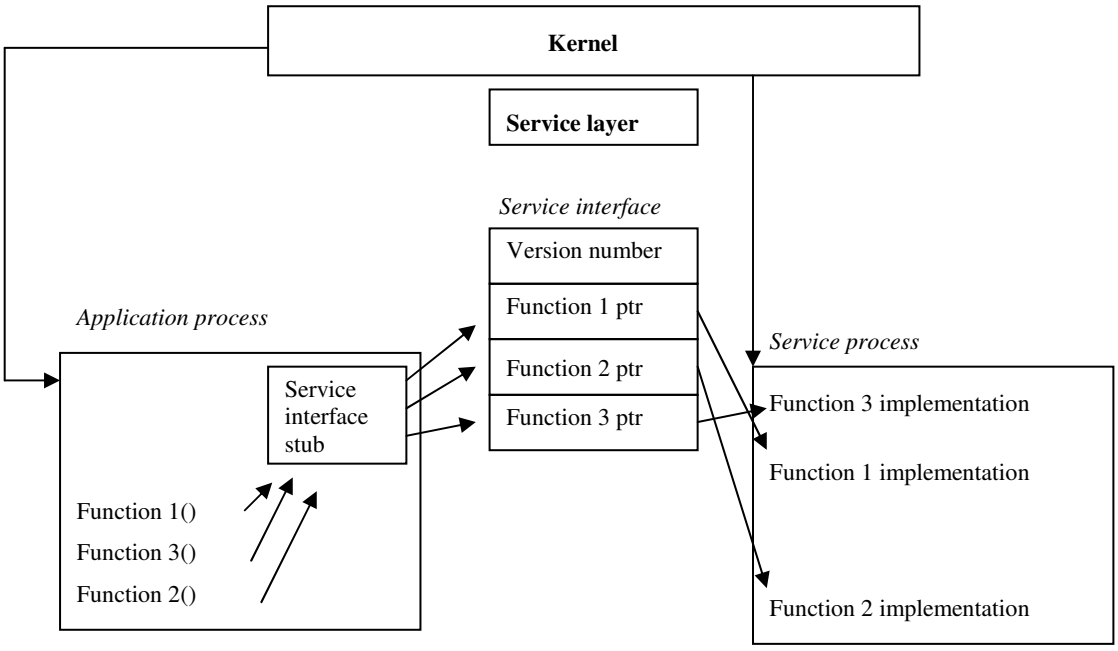


Figura 3.2 Services

Come gli altri processi, anche i servizi possono essere caricati dinamicamente e rimpiazzati. E' cruciale che il process ID venga mantenuto, dato che è utilizzato per l'identificazione del service.
Quando un service deve essere rimpiazzato, il kernel informa la versione in esecuzione inviando un evento particolare al service process. Il service prima di rimuoversi autonomamente dal sistema deve trasferire l'internal state al nuovo service, tramite passaggio di puntatore e la memoria deve essere prelevata da una zona condivisa, dato che la memoria del vecchio processo viene completamente liberata. Nella descrizione del stato del service è marchiata con il numero di versione del service, in modo che versioni incompatibili dello stesso servizio non tenteranno di caricare la descrizione del service.

## 3.7 Libraries

Il kernel di Contiki fornisce le sole funzionalità base di multiplexing della CPU e la gestione degli eventi, mentre il resto del sistema è implementato con librerie di sistema che vengono opzionalmente linkate nei programmi in tre modi.
Nel primo modo i programmi possono essere staticamente linkati alle librerie che fanno parte del core, nel secondo modo i programmi possono essere linkati con librerie che fanno parte dei programmi caricabili e nel terzo modo i programmi possono richiamare services che implementano una libreria specifica.
Le librerie implementate come servizi possono essere rimpiazzate dinamicamente durante il run-time.
Le librerie che fanno parte del core sono sempre presenti nel sistema e non devono essere incluse nei binari dei programmi caricabili.

## 3.8 Communication support

La comunicazione è fondamentale nelle reti di sensori ed è implementata in Contiki come service, per consentirne la sostituzione durante il run-time e per permettere di caricarne contemporaneamente più di uno.
Inoltre lo stack può essere splittato in più componenti per consentire la sostituzione di singole componenti. I service di comunicazione utilizzano il meccanismo dei service per richiamarsi tra loro e gli eventi sincroni per comunicare con gli applicativi.
Gli eventi sincroni devono arrivare a completamento e perciò si può utilizzare un unico buffer per i processi di comunicazione, senza necessità di copiare dati.

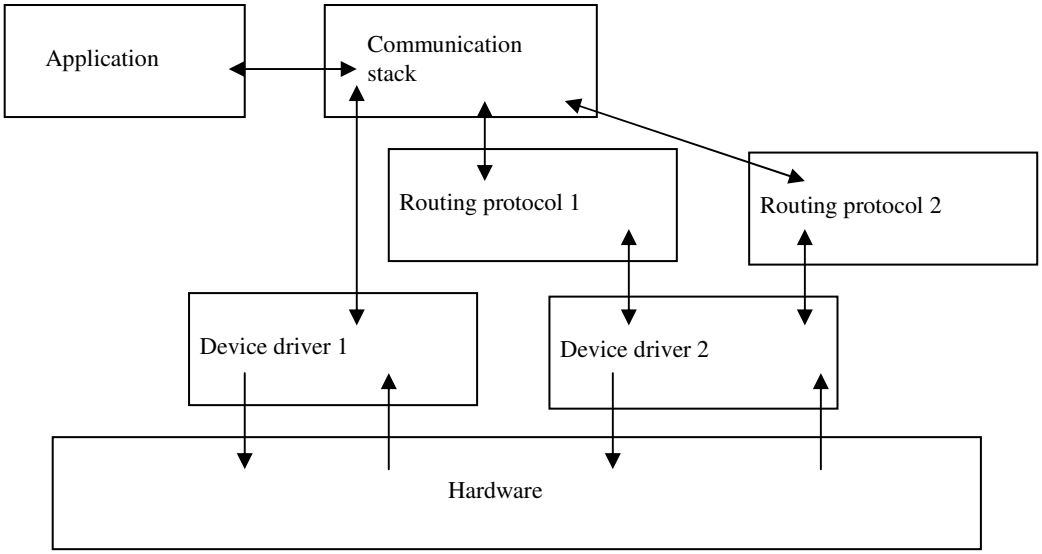


Figura 3.3 Communication support

Un device driver legge i pacchetti in ingresso nel comunication buffer e poi richiama i service dei livelli superiori con il meccanismo dei service. Il communication stack elabora l'header del pacchetto e genera un evento sincrono all'applicazione destinataria del pacchetto. L'applicazione lavora sui contenuti del pacchetto ed eventualmente posiziona una risposta nel buffer, prima di restituire il controllo al comunication stack, che aggiunge i propri header al pacchetto in uscita e restituisce il controllo al device driver, in modo da consentirne la trasmissione.

## 3.9 uIPv6

La maggior parte dei sensori wireless che utilizzano IEEE 802.15.4 come layer fisico e MAC, hanno stack di comunicazione proprietari o definiti in alleanze esclusive tra vendor, come ad esempio Z-Wave o Zigbee, rendendo difficile il raggiungimento dell'interoperabilità.
L'utilizzo di IP nel layer 3 è arenato vista la complessità che ostacola la portabilità in un dispositivo con risorse limitate.
Lo stack IPv6 per dispositivi con memoria limitata è detto uIPv6 ed è il primo stack ad aver superato tutti i test della fase 1 della certificazione IPv6 ready.
Contiki implementa l'header compression e fragmentation sulla base della RFC 4944 e la versione 01 del nuovo meccanismo di header compression context-based.
Contiki non implementa layer 2 mesh feature (mesh header, bcast header), dato che si focalizza sul routing a livello 3.
La dimensione di codice in Contiki 2.3 per uIPv6 è di 11,5 Kbytes e richiede meno di 2 Kbytes.

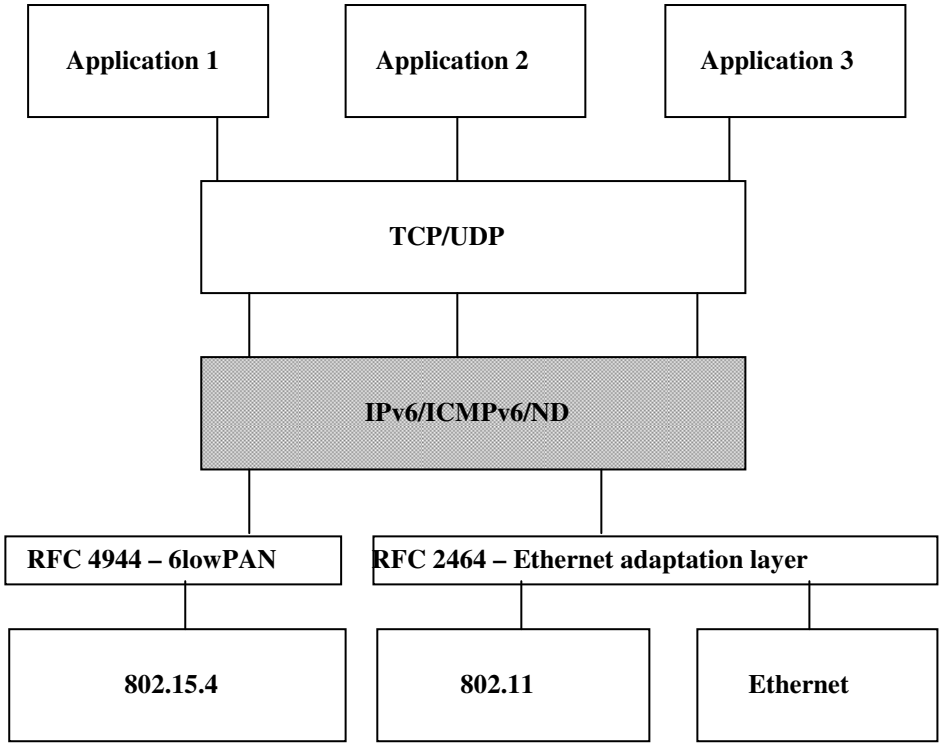


Figura 3.4 uIPv6

uIPv6 non dipende da particolari MAC layer o layer fisici e l'interfaccia è composta da due wrappers, per le funzioni di in/out, l'indirizzo del link-layer e alcune costanti. Questo crea un livello di astrazione, che permette l'integrazione di diversi MAC e protocolli di link layer.
In Contiki, allo start-up del sistema, uIPv6 esegue le seguenti operazioni:

1. inizializza l'interfaccia di rete.
2. il nodo crea il proprio indirizzo link-local IPv6, associando il prefisso fe80 :: 0/64 al proprio MAC address 802.15.4.
3. effettua in Duplicate Address Detection (DAD), per verificare che nessun altro nodo abbia lo stesso indirizzo.
4. in parallelo il nodo emette un router solicitation message per ricevere advertisement dal router della rete.
5. il nodo utilizza le informazioni ricevute per creare l'indirizzo globale ed aggiornare i proprio parametri.

Quando un nodo deve inviare un pacchetto, il nodo effettua una next-hop determination per trovare a quale nodo confinante inviare il pacchetto. Se il MAC address del confinante non è nella cache, il nodo esegue una risoluzione dell'indirizzo.
Lo stack uIPv6 utilizza un solo buffer globale per i pacchetti entranti ed uscenti, la cui lunghezza corrisponde alla lunghezza del header MAC più 1280 bytes (minima MTU link).
Buffers addizionali sono disponibili per le operazioni di fragment reassembly buffering per nodo confinante.
Le strutture principali sono: interface address list, neighbor cache, prefix list e default router list, richiesti da ND.

Lo stack IPv6 utilizza 2 timer periodici per rimuovere le informazioni obsolete da queste strutture.
Di seguito il footprint di Contiki 2.1 per il codice e memoria per l'uIPv6 stack, in bytes, compilato per Atmel RAVEN, con Atmega 1248P MCU con 128 kbytes di flash e 16 kbytes di RAM, compilato con avr-gcc-4.2.2.

| **Function** | **ROM** | **RAM** |
|---|---:|---:|
| ND Input/Output | 4800 | 20 |
| ND structures | 2128 | 238 |
| Network interface management | 1348 | 118 |
| Stateless address autoconf | 372 | 16 |
| IPv6 (header processing, etc.) | 1434 | 44 |
| Packet buffer | 0 | 1296 |
| ICMPv6 | 1406 | 16 |
| **Total** | **11488** | **1748** |

Un sistema completo prevede una dimensione del codice totale di 35 KB, comprensivi dei driver RAVEN, 802.15.4 PHY e MAC, 6lowPAN con frammentazione e compressione dell'header, uIPv6 ed il sistema Contiki. UDP richiede altri 1.3 K mentre TCP richiede altri 4 KB.

## 3.10 Implementazione 6LoWPAN

L'implementazione 6LoWPAN di Contiki si basa su RFC 4944 Trasmission of IPv6 Packets over IEEE 802.15.4 Networks, draft-hui-6lowpan-interop-00 Interoperability Test for 6LoWPAN e draft-hui-6lowpan-hc-01 Compression format for IPv6 datagrams in 6lowpan Networks.
Contiki implementa l'indirizzamento a 64 bit, la frammentazione e la compressione dell'header stateless e stateful. Contiki non sopporta le modalità mesh under e si focalizza su tecniche di route over.
6LoWPAN non è implementato come processo ma viene chiamato dal processo MAC quando riceve un pacchetto 6lowpan e dal processo tcpip quando è necessario inviare un pacchetto IPv6.
Il processo MAC inizializza 6lowpan chiamando sicslowpan_init, con un puntatore alla struttura mac_driver come argomento.
Le funzioni 6lowpan sono implementate in sicslowpan.h e sicslowpan.c che vengono utilizzate per gestire i pacchetti tra il layer 802.15.4 e IPv6.
In fase di inizializzazione, la funzione di input in sicslowpan.c viene impostata come funzione da richiamare dal MAC alla ricezione del pacchetto e la funzione di output di sicslopan.c è impostata come funzione tcpip_output.
Alla ricezione del pacchetto il link-layer copia il payload 802.15.4 nel buffer rime e ne imposta la lunghezza, ricava l'indirizzo di source e destination link-layer come indirizzi RIME e poi viene chiamata la funzione di input siclowpan.
Analogamente quando il layer IPv6 deve inviare tramite radio un pacchetto, lo posiziona in uip_buf, ne imposta la lunghezza e chiama la funzione di output di sicslowpan.

Per quanto riguarda la frammentazione, viene applicata dalla funzione di output solo quando un pacchetto IPv6, anche dopo la compressione dell'header, risulta troppo grande per essere trasportato da una frame 802.15.4: il pacchetto viene diviso in più pacchetti secondo RFC 4944 e solo il primo contiene i campi dell'header IP/UDP. La funzione di input invece si occupa della ricostruzione a partire dai frammenti che non giungono sempre in ordine e che devono essere ricevuti entro il tempo massimo di 20 secondi di default. Durante il processo di ricostruzione del pacchetto vengono scartati i frammenti appartenenti ad altri pacchetti.
Alla ricezione del primo frammento viene definito un nuovo buffer, sicslowpan_buf di 1280 byte per riassemblare il pacchetto e quando la ricostruzione è completa il pacchetto viene copiato in uip_buf, viene impostato il valore della lunghezza in uip_len e poi viene chiamata la funzione tcpip_input.
Dal punto di vista della compressione dell'header, l'implementazione Contiki di 6LoWPAN supporta le compressioni stateless HC1, stateful HC01 ed IPv6.
La compressione IPv6 indica l'invio del pacchetto senza compressione, con il dispatch IPv6 prima dell'header IPv6.
Se in fase di compilazione viene scelta la compressione IPv6, i pacchetti inviati non verranno mai compressi ed i pacchetti compressi ricevuti non verranno elaborati.
Se in fase di compilazione viene scelta la compressione HC1 o HC01, si utilizzarenno le tecniche di compressioni in fase di output e ogni tipo di compressione in input verrà elaborata, compresa la versione IPv6. In fase di ricezione, viene chiamata la funzione di input, viene gestita la fase di ricostruzione del pacchetto e poi viene verificato il byte dispatch: se indica IPv6, viene gestito direttamente, mentre se è HC1 o HC01 viene chiamata la relativa funzione.
In fase di trasmissione, se la compressione è IPv6, viene solo inserito il dispatch corrispondente, altrimenti vengono chiamate le funzioni di compressione.
In HC1 possono essere compressi solo indirizzi unicast link address, il pacchetto viene compresso solo se tutti i campi possono essere compressi (flow label compreso), altrimenti viene inviato il pacchetto inline.
HC01 utilizza indirizzi di context per la compressione di indirizzi unicast globali e tutti i nodi devono condividere le stesse informazioni di context, che devono essere distribuite tramite estensioni di ND.

## 3.11 Preemptive multi-threading

Preemptive multi-threading è implementato in Contiki come libreria sopra al kernel che può essere linkata opzionalmente alle applicazioni che lo prevedono esplicitamente. La libreira è divisa in due parti: una parte indipendente dalla piattaforma, che si interfaccia con il kernel ed una parte specifica per piattaforma che implementa lo switching dello stack e le primitive di preemtion (25 linee di codice C per MSP430).
La preemption è implementata utilizzando un timer interrupt che salva i valori dei registri del processore nello stack e ritorna allo stack del kernel. Ogni thread, a differenza dei normali processi, richiedono uno stack separato ed ogni thread esegue il proprio stack finchè lo rilasciano oppure vengono preempted.

## 3.12 Programmazione tramite wireless

La programmazione tramite wireless avviene mediante la trasmissione di un unico file binario al nodo concentratore, utilizzando una comunicazione point-to-point. Il file binario viene salvato in EEPROM, quando il programma è stato completamente trasmesso e successivamente viene inviato in broadcast ai nodi confinanti. Eventuali perdite di pacchetti vengono comunicate dai nodi confinanti tramite negative ack e la ritrasmissione viene fatta dal nodo concentratore.

## 3.13 Organizzazione e dimensione del codice

Un sistema operativo per sistemi limitati deve essere compatto sia in ternimi di dimensione del codice che in termini di occupazione della RAM, per lasciare spazio alle applicazioni. La seguente tabella evidenzia i dati ricavati dall'analisi della dimensione del codice e l'occupazione della RAM da parte di Contiki in versione 2.1, con un interessante legame con il numero di processi ed eventi nel caso delle due piattaforme più diffuse TI MSP430 e Atmel AVR.
Oltre alle componenti di Contiki si evidenzia l'occupazione da parte dell'applicazione di replica dei dati di un sensore, composto da un service stub e dal service stesso. Il program loader è implementato solo per l'MSP430.

| **Module** | **Code size (AVR)** | **Code size (MSP430)** | **RAM usage** |
|---|---|---|---|
| kernel | 1044 | 810 | 10+4e+2p |
| Service layer | 128 | 110 | 0 |
| Program loader | - | 658 | 8 |
| Multi-threading | 678 | 582 | 8+s |
| Timer library | 90 | 60 | 0 |
| Replicator stub | 182 | 98 | 4 |
| Replicator | 1752 | 1558 | 200 |
| **Total** | **3874** | **3876** | **230+4e+2p+s** |

La quantità di RAM richiesta dipende dal massimo numero di processi configurati per il sistema (p), la massima dimensione della coda asincrona degli eventi (e) ed in caso di multi-threading, la dimensione dello stack del thread (s).
Le dimensioni dei moduli possono essere ricavate mediante l'analisi del file .map prodotto dal linker, oppure tramite il tool *objdump*, che nel caso della piattaforma SKY/Telosb, viene realizzato da *msp430-objdump*.
In appendice A viene riportato l'elenco dei moduli, con relativi file e funzioni della versione Contiki 2.3 utilizzata in questo lavoro, mentre in appendice B viene riportata l'analisi delle dimensioni dei moduli eseguita su un'applicazione di Contiki 2.3.

# 4. Contiki 2.3

## 4.1 Instant Contiki

Contiki nella release 2.3 è disponibile in versione virtual machine VMware, denominata Instant Contiki per la completezza delle risorse fornite per lo sviluppo oppure nella versione source code, che richiede l'utilizzo di un compilatore C in ambito Linux o Windows.
La versione Instant Contiki può essere prelevata dal sito del Swedish Institute of Computer Science (SICS) e richiede l'installazione di VMware Player.
Instant Contiki si basa su Linux Ubuntu e contiene i compilatori per i processori dei sensori supportati, tra i quali MSP430 e AVR.
Dopo aver installato il driver FTDI per Tmote Sky è possibile connettere il sensore tramite USB, che sarà riconosciuto da VMware Player come "Future Technology Device". Eventuali incompatibilità saranno risolte aggiornado Linux Ubuntu con le patch disponibili.

## 4.2 Operazioni di fix iniziale e verifica connettività

Dopo aver installato Instant Contiki è necessario verificare le proprietà di scrittua dei file di Contiki, per consentire la compilazione e caricamento degli eseguibili nei sensori.
Per verificare la connettività USB e la programmazione di un sensore è possibile utilizzare l'applicazione blink che accende i vari LED del sensore.
Per rimuovere il sensore è necessario disattivarlo da VMware Player, tramite le icone dei device connessi alle porte USB.

## 4.3 Operazioni di maintenance e programmazione

L'ambiente di sviluppo Instant Contiki propone la versione 2.3 di Contiki e la versione 2.x in continuo sviluppo.
L'aggiornamento dell'ambiente Contiki 2.x può essere eseguito on-line tramite il comand cvs, partendo dalla home dell'utente user.
L'ambiente di sviluppo propone vari esempi ordinati in folder separati e da ognuno di essi è possibile effettuare la verifica del numero e della tipologia dei sensori connessi tramite i seguenti argomenti del comando *make*:

```
make TARGET=sky sky-motelist
```

La connessione ad ogni dispositivo può avvenire dal Linux, tramite i seguenti parametri del comando *make*:

```
make login MOTE=x
```

con x=1 per /dev/ttyUSB0 e x=2 per /dev/ttyUSB1.
La programmazione del singolo sensore avviene utilizzando il seguente comando:

```
make TARGET=sky nomefile.upload MOTE=x
```

con x=1 per /dev/ttyUSB0 e x=2 per /dev/ttyUSB1 e nomefile.upload rappresenta la versione dell'eseguibile compilata per il TARGET specificato, nel nostro caso sky.

## 4.4 Simulazione in MSPSim

MSP430 è un emulatore Java a livello di istruzioni, per microprocessori della serie MSP430 e di alcune piattaforme di sensori di rete.
MSPSim supporta il caricamento di firmware IHEX e ELF ed ha alcuni tool per il monitoraggio dello stack, il settaggio di breakpoint e profiling e può lavorare autonomamente o in combinazione con COOJA, nel quale emula il nodo mentre COOJA fornisce la simulazione del mezzo trasmissivo ed altre feature.
La versione standalone di MSPSim è utile per lo sviluppo del codice del singolo nodo, ad esempio per lo sviluppo del filesystem Coffe, in combinazione con la shell.
MSSim può essere lanciato direttamente compilando una qualsiasi applicazione con l'estensione *mspsim*.
Il commando lancia automaticamente MSPSim e in una finestra "USRT1 Port Output" verranno evidenziate le linee prodotte in fase di boot-up del codice.
MSPSim include anche i LEDs che blinkano e la possibilità di simulare il tasto presente nel sensore.

## 4.5 Ambiente di simulazione COOJA

Cooja [8] si basa su Java, è contenuto in Instant Contiki e permette di simulare il software destinato ai sensori in un ambiente network-based. In Cooja i nodi possono essere simulati oppure emulati: nella prima metodologia i nodi simulati vergono compilati ed eseguiti nativamente mentre nella seconda metodologia, che richiede più risorse, i nodi utilizzano MSPSim, l'emulatore MSP430, per caricare ed eseguire il firmware.
Per utilizzare Cooja, è necessario compilarlo ed eseguirlo, mediante l'istruzione *ant run*. Dopo la compilazione, viene eseguito Cooja senza simulazione nè i plug-in che interagiscono con la simulazione e ricreano i nodi.
La simulazione può essere create tramite la voce di menù File-New Simulation, selezionando le opzioni richieste.
Per aggiungere un nodo alla simulazione è necessario che venga prima creato tramite la voce di menù Mote Types-Create mote type-Contiki Mote Type e poi compilato.
Dopo la creazione di un nodo è possibile aggiungere uno o più nodi alla simulazione, tramite la voce di menù Motes-Add notes of type, indicando la quantitù di nodi.
La simulazione può iniziare con due plug-in: un log listener che ascolta le porte seriali di tutti i nodi ed un visualizer indica le informazioni in merito ai nodi. Sono disponibili altri plug-in, come ad esempio UDGM visualizer che permette di analizzare le trasmissioni radio e di modificarne il range.
Per far iniziare la simulazione si può utilizzare Start dal Control Panel inoltre è possibile salvare la configurazione della simulazione, tramite il menù File-Save simulation. Le simulazioni vengono salvate in file di estensione .csc, caricabili

successivamente tramite la voce di menù File-Open simulation-Browse. Al caricamento della simulazione, tutte le applicazioni Contiki vengono ricompilate.

## 4.6 Programmazione tramite wireless

La programmazione tramite wireless, detta code dissemination, permette di posizionare il codice in un file e successivamente chiamare la funzione elfloader_load(fd), dove fd è la descrizione del file che è restituita aprendo il file tramite cfs_open().
La funzione elfloader_load(fd) carica un file ELF nel file system. Il progetto si chiama "Deluge" ed è disponibile un test con l'applicazione test-deluge.c

## 4.7 Alcuni dati sui programmi Contiki

Di seguito alcuni dati di occupazione EEPROM (text) e RAM (data+bss) di alcuni programmi caricati nella versione Contiki 2.3.
I dati riportati sono stati estratti tramite il comando "msp430-size nome binary.sky".

| **name** | **text** | **data** | **bss** | **dec** |
|---|---|---|---|---|
| **Software di base** | | | | |
| hello-world.sky | 23116 | 190 | 5119 | 28425 |
| sky-shell | 48080 | 1064 | 6133 | 55277 |
| **IPv4 example** | | | | |
| sky-webserver.sky | 42352 | 284 | 7133 | 49769 |
| **IPv6 tools** | | | | |
| uip6-bridge-tap.sky | 24772 | 160 | 5952 | 30884 |
| **IPv6 example** | | | | |
| empty.sky | 39848 | 150 | 7422 | 47322 |
| example-ping6 | - | - | - | - |
| testv6 | - | - | - | - |
| udp-client (in udp-ipv6) | 41344 | 150 | 7428 | 48922 |
| udp-server (in udp-ipv6) | 41222 | 150 | 7420 | 48792 |
| example-udp-receiver-ipv6 | 41064 | 150 | 7416 | 48630 |
| example-udp-sender-ipv6 | 41148 | 152 | 7424 | 48724 |

Per ottenere un dettaglio si può analizzare il file .map generato in fase di compilazione, si possono inserire linker switches per ottenere più dettagli, oppure si può utilizzare il comando objdump per l'analisi del file compilato.
In appendice D si ripota l'elenco dei comandi disponibili con l'applicazione shell.

## 4.8 Hello-world

Di seguito si riporta il codice dell' applicazione Hello-world.c per Contiki.

```
#include “contiki.h”

#include <stdio.h> /*for printf() */
/* Declare the process */
PROCESS(hello_world_process, “Hello world”);
/* Make the process start when the module is loaded */
AUTOSTART_PROCESSES(&hello_world_process);
/* Define the process code */
PROCESS_THREAD(hello_world_process, ev, data) {
PROCESS_BEGIN(); /* Must always come first */
printf(“Hello, world!\n”); /* Initialization code goes here */
while(1) { /* Loop for ever */
PROCESS_WAIT_EVENT(); /* Wait for something to happen */
}
PROCESS_END(); /* Must always come last */
}
```

Da questo semplice codice si possono ricavare gli elementi fondamentali di un codice per Contiki e nell’ordine richiesto: prima di tutto è importante dichiarare un processo con l’istruzione PROCESS() e farlo attivare al caricamento del software, tramite l’istruzione AUTOSTART().
Si passa poi alla definizione del codice con PROCESS_THREAD(), che all’interno contiene PROCESS_BEGIN() e PROCESS_END().
Tra PROCESS_BEGIN() e PROCESS_END(), troviamo la vera parte di codice, preceduta da una zona di inizializzazione.
Questo codice può essere simulato direttamente nell’ambiente di sviluppo con il target native oppure può essere caricato nel sensore e si otterrà un messaggio in console.

## 4.9 IPv6 empty

Di seguito si riporta il codice dell’ applicazione empty.c per Contiki.

```
#include "contiki.h"
//#include <stdio.h>

/*---------------------------------------------------------------------------*/
PROCESS(init_stack_process, "Init stack process");
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(init_stack_process, ev, data)
{
  PROCESS_BEGIN();
  //printf("WARNING: Running Contiki without application\n");
  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
AUTOSTART_PROCESSES(&init_stack_process);
/*---------------------------------------------------------------------------*/
```

Questa applicazione è costituita dal sistema operativo e dalla componente uIP con adaptation layer 6LoWPAN ed è adatta per l’analisi approfondita delle componenti di Contiki.
L’analisi è stata effettuata mediante il file map generato dal linker e dai dati prodotti dal tool objdump della versione 2.3 del codice empty.sky.

Si riportano in appendice B i dati di estensione delle singole componenti nella EEPROM e nella RAM del device e di seguito i dati comulativi delle estensioni dei moduli del sistema operativo.
Di seguito si riportano i dati aggregati per componente:

| COMPONENTE | TEXT length (ROM) | DATA length (RAM) | BSS length (RAM) | COMMON (RAM) |
|---|---|---|---|---|
| **Library: contiki** | **1350** | **0** | **0** | **0** |
| **Library: external** | **4880** | **6** | **6** | **0** |
| **networking: IPv6** | **17158** | **13** | **1472** | **949** |
| **networking: mac** | **3834** | **30** | **193** | **2** |
| **networking: rime** | **2022** | **36** | **4317** | **155** |
| **platform** | **1686** | **4** | **75** | **0** |
| **system: base (core)** | **4130** | **46** | **108** | **102** |
| **system: device drivers** | **4756** | **14** | **18** | **18** |

Di seguito viene riportata la distribuzione del codice in ROM:

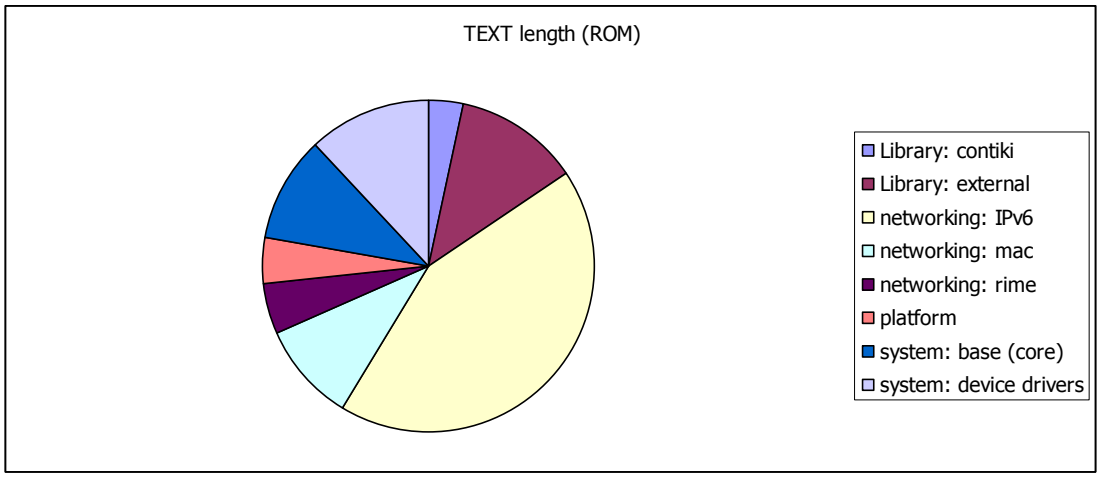


Figura 4.1 Organizzazione della ROM di empty

Di seguito viene riportata la distribuzione del codice in RAM:

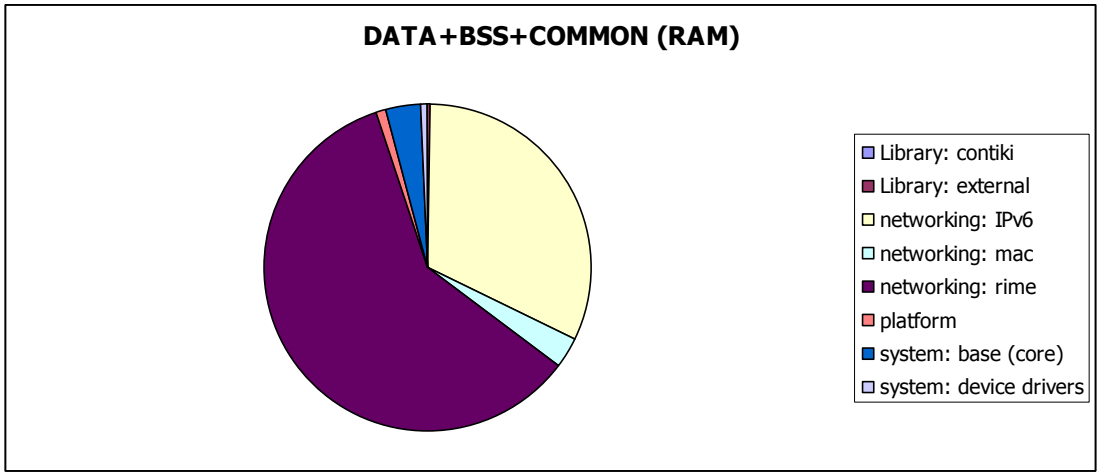


Figura 4.2 Organizzazione delle RAM di empty

Di seguito altri dati relativi all'organizzazione della memoria:

**Memory Configuration**

| Name | Origin | Length |
|---|---|---|
| text | 0x00004000 | 0x0000bfe0 |
| data | 0x00001100 | 0x00002800 |
| vectors | 0x0000ffe0 | 0x00000020 |
| bootloader | 0x00000c00 | 0x00000400 |
| infomem | 0x00001000 | 0x00000100 |
| infomemnobits | 0x00001000 | 0x00000100 |

## 4.10 IPv6 Bridge

L'applicazione uip6-bridge-tap consente di trasformare il Linux con un mote connesso tramite USB in un bridge tra lo stub network 6LoWPAN ed Internet, con limitate funzioni di edge routing.
Per verificare le funzionalità di uip-6-bridge, è stata scelta l'applicazione empty nel secondo nodo.
Per trasformare il Linux in un bridge, prima di tutto è necessario arricchire Linux di alcune funzionalità necessarie per erogare servizi allo stub network, come il comando service ed il pacchetto radvd.
Per la configurazione di radvd, è importante posizionare in /etc una copia di radvd.conf con definito il prefix.
Si procede con la compilazione di uip6-bridge e si esegue l'upload nel primo sensore, uip-bridge-tap.sky:
Per quanto riguarda il secondo nodo, si compila empty per il secondo sensore e si esegue l'upload nel secondo sensore.
A questo punto, si può procedere all'attivazione del canale slip su tap0 tra Linux ed il primo mote con uip-bridge.
Dopo aver creato il tap0, si popola la tabella di routing IPv6 con l'extension attribuito dal servizio radvd allo stub 6lowpan e si riavvia il servizio stesso: a questo punto si può procedere al test da Linux, eseguendo un ping6 verso l'indirizzo global del secondo mote.
Di seguito si riportano i dati di performance:

```
1387 packets transmitted, 1140 received, +30 errors, 17% packet loss, time 1387240ms
rtt min/avg/max/mdev = 63.812/629.421/2517.245/298.209 ms, pipe 3
```

Lo sniffing del raffico icmp può essere effettuato mediante cattura sull'interfaccia tap0, che fornirà solo traffico IPv6, con tcpdump, ma si otterranno pacchetti privi degli header 6lowpan.
Di seguito la cattura effettuata tramite tcpdump su tap0, visualizzata tramite Wireshark:

| | | | | |
|---|---|---|---|---|
| 33 17.091507 | fe80::7600:12ff:fee5:e717 | aaaa::212:7400:137b:ea2a | ICMPv6 | Echo request |
| 34 17.999121 | aaaa::212:7400:137b:ea2a | fe80::7600:12ff:fee5:e717 | ICMPv6 | Echo reply |
| 35 18.090540 | fe80::7600:12ff:fee5:e717 | aaaa::212:7400:137b:ea2a | ICMPv6 | Echo request |
| 36 19.006318 | aaaa::212:7400:137b:ea2a | fe80::7600:12ff:fee5:e717 | ICMPv6 | Echo reply |
| 37 19.089581 | fe80::7600:12ff:fee5:e717 | aaaa::212:7400:137b:ea2a | ICMPv6 | Echo request |
| 38 20.000943 | aaaa::212:7400:137b:ea2a | fe80::7600:12ff:fee5:e717 | ICMPv6 | Echo reply |

Figura 4.3 Contiki ICMPv6 capture

Di seguito l'analisi di un singolo pacchetto della cattura effettuata tramite tcpdump su tap0, visualizzata tramite Wireshark:

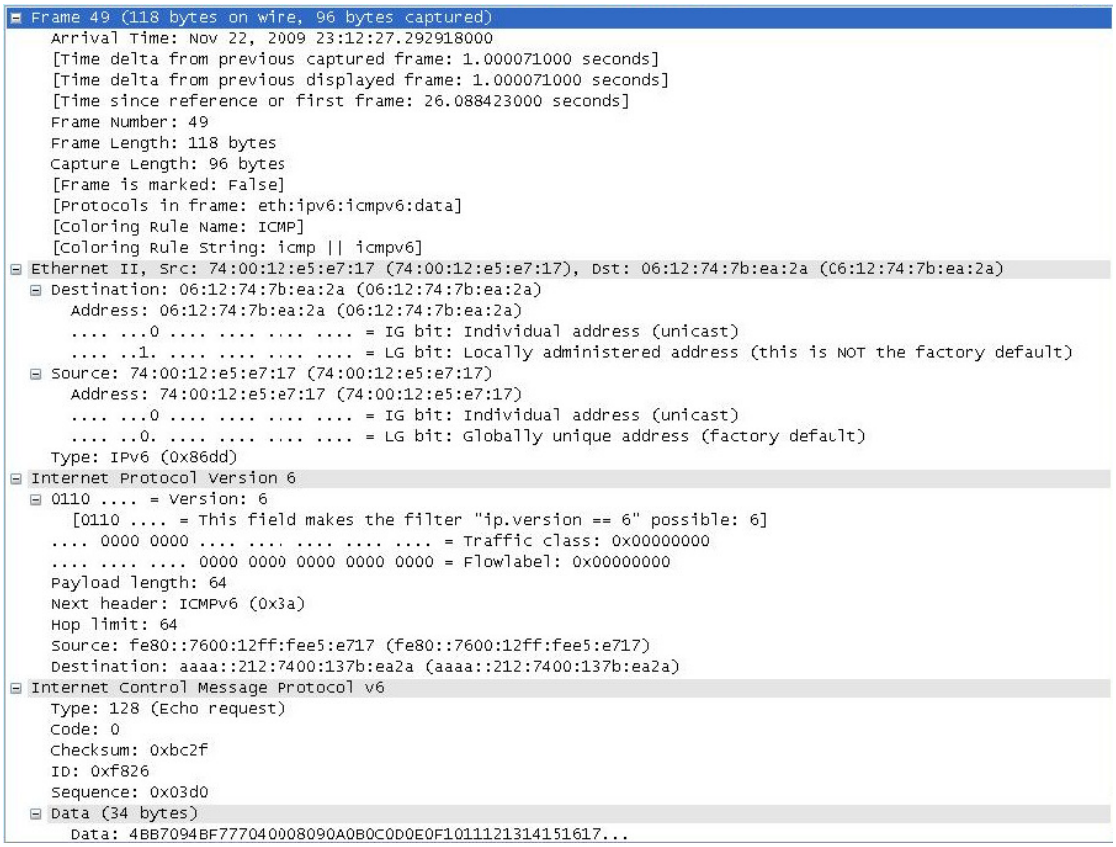


Figura 4.4 Contiki ICMPv6 packet

## 4.11 IPv6 Ping

Contiki 2.3 mette a disposizione l'applicazione per l'invio di pacchetti icmp echo verso una destinazione definita in configurazione.
L'applicazione è definita tramite il seguente codice:

```
#include "ping6.h"

/*---------------------------------------------------------------------------*/
AUTOSTART_PROCESSES(&ping6_process);
/*---------------------------------------------------------------------------*/
```

La definizione manuale dell'indirizzo della destinazione dei pacchetti icmp echo si inserisce nel file ping6.c e si attiva tramite il parametro MACDEBUG.

## 4.12 IPv6 UDP Sender e Receiver

L'applicazione consente di inviare dal mote udp-sender-ipv6 una stringa al mote con udp-reveicer-ipv6 e di ricevere una risposta di conferma tramite protocollo UDP, visualizzabili da console dei mote.

In particolare nel mote con udp-sender-ipv6 è necessario impostare nel file example-udp-sender.c alcuni parametri: prima di tutto è necessario definire la stringa ed impostare l'indirizzo di destinazione del pacchetto UDP e la porta di destinazione e source del pacchetto UDP.
Nel mote con udp-receiver-ipv6 è necessario impostare nel file example-udp-receiver.c i parametri speculari rispetto all'altro mote, in particolare l'ip di source e le porte udp di origine e destinazione.

# 5. Test di compatibilità 6lowPAN

## 5.1 Criteri per compatibilità IPv6

L'aderenza allo standard è essenziale per assicurare l'interoperabilità, come indicato dalla RFC 4294 [2] per host e router in IPv6. In particolare è richiesto che un nodo IPv6 rispetti con i seguenti standard: IPv6 specification, IPv6 Addressing Architecture, Neighbor Discovery, ICMP per IPv6, Stateless address configuration, Default Address Selection e Multicast Listener Discovery.
Per assicurare la compatibilità con lo standard, l'IPv6 Forum ha creato il programma "IPv6 Ready" che restituisce la certificazione per i due livelli Phase-1 e Phase-2, a seguito di test rigorosi di interoperabilità.
Il sistema operativo Contiki 2.3 ha raggiunto la certificazione "IPv6 Ready" a fine 2008 [7] mentre il sistema operativo TinyOS non ha superato tale certificazione.

## 5.2 Criteri per l'interoperabilità 6LoWPAN

Per determinare l'interoperabilità tra dispositivi tramite 6loWPAN è necessario che almeno due realizzazioni indipendenti siano interoperabili.
6loWPAN supporta un approccio pay-as-you-go, nel quale i pacchetti utilizzati più frequentemente sono altamente compressi, mentre i pacchetti meno utilizzati sono inseriti come sono.
La completa interoperabilità richiede la verifica con tutte le combinazioni dei formati 6loWPAN, iniziamo con alcuni test per dimostrarne piccole parti, definendo il livello 0 ed il livello 1 di compatibilità.
Il livello 0 prevede la comunicazione unicast compressa IPHC, tramite icmp mentre il livello 1 prevede la comunicazione unicast IPHC, con scambio di pacchetto UDP.
Il livello 2 prevede la comunicazione unicast compressa IPHC tramite pacchetti UDP con frammentazione.

## 5.3 Level 0 interoperability

L'interoperabilità di livello 0 deve dimostrare la capacità di generare e processare pacchetti unicast compressi IPHC, utilizzano il protocollo ICMPv6, con pacchetti echo request/reply.
Il level 0 non dimostra la capacità di generare e processare 6loWPAN Mesh, Fragmentation e broadcast headers.

## 5.4 Level 1 interoperability

Il livello 1 deve dimostrare la capacità di entrambi i nodi di generare e processare pacchetti unicast compressi IPHC, utilizzando il protocollo UDP con compressione LOWPAN_NHC e LOWPAN_UDP.

Il level 1 non dimostra la capacità di generare e processare 6loWPAN Mesh, Fragmentation e broadcast headers.
Per i test di interoperabilità di livello 1 in sintesi si deve verificare la capacità di gestire l'header fully-compressed, cioè il minimo formato di header che supporti le funzionalità richieste.
Nel caso di unicast link-local, l'header IPv6 è ridotto a 2 otteti: hop limit (1 otteto) e LOWPAN_HC1 Encoding (1 otteto).
Per quanto riguarda i campi compressi di IPv6, version è sempre IPv6, traffic class e flow label è 1, payload length ed entrambi gli indirizzi sono derivati dall'header 802.15.4, next header è UDP.

## 5.5 Level 2 interoperability

Il livello 2 deve dimostrare la capacità di entrambi i nodi di generare e processare pacchetti unicast compressi IPHC, utilizzando il protocollo UDP con compressione LOWPAN_NHC e LOWPAN_UDP, con payload che eccede la capacità massima della singola frame.
Il level 2 non dimostra la capacità di generare e processare 6loWPAN Mesh e broadcast headers.
Per i test di interoperabilità di livello 2 in sintesi si deve verificare la capacità di gestire la frammentazione in aggiunta alle funzioni di compressione dell'header.

## 5.6 Preparazione ai test di interoperabilità

Da un confronto approfondito tra i sistemi Contiki 2.3 e TinyOS 2.x sono emerse le seguenti differenze:

1- Il software blip v. 1.0 di TinyOS non supporta l'indirizzamento IEEE 802.15.4 a 64 bit e comunque nella versione IEEE 802.15.4 con indirizzamento 16 bit non richiede la presenza di un PAN coordinator o edge router, ma consente la configurazione manuale dell'indirizzamento, tramite parametri dati in fase di compilazione.
2- L'header compression stateful IPHC di 6lowpan di TinyOS si basa sul draft-hui-6lowpan-hc-00 mentre nella versione 2.3 di Contiki si utilizza il draft-hui-6lowpan-hc-01 [5]. L'attuale versione 2.4 di Contiki utilizza la versione draft-ietf-6lowpan-hc-06.
3- Contiki adotta l'RFC2461 IPv6 ND e perciò non utilizza funzioni di registrazione all'edge router, definite nei draft-ietf-6lowpan-nd.
4- Il software blip v. 1.0 di TinyOS non utilizza il protocollo ND per l'interazione con i confinanti ed invia direttamente la frame all'indirizzo link-layer ricavato dall'indirizzo IPv6.
5- L'associazione tra l'indirizzo link-layer a l'indirizzo IPv6 in TinyOS prevede bit con valore 0 nelle posizioni comprese tra 17 e 64, a differenza del RFC 4944 che prevede alcuni bit di valore 1 e alcuni bit con valore ricavato dal PAN-Id.
6- Il primo bit di LOWPAN_UDP in TinyOS viene settato ad 1 anche in caso di compressione, mentre il draft-hui-6lowpan-hc-00 richiederebbe il bit con valore 0.

Queste peculiarità hanno imposto l'adeguamento di uno dei due sistemi operativi, in questo caso Contiki 2.3.

### 5.6.1 Attivazione indirizzamento 802.15.4 su Contiki 2.3

Per attivare l'utilizzo dell'indirizzamento IEEE 802.15.4 a 16 bit sono stati modificati alcuni parametri ed sono state aggiunte alcune righe di codice, partendo dalla versione Contiki 2.x, basata sulla versione 2.4, oggetto di sviluppo continuo ed aggiornabile tramite cvs.
Prima di tutto si parte dai parametri RIME, dato che l'indirizzamento link layer deriva ed utilizza alcune funzione del precedente sistema di indirizzamento non IP.
Come prima modifica l'indirizzamento RIME è stato portato da 8 byte a 2 byte, modificando il file contiki-conf.h.
L'indirizzamento IEEE 802.15.4 a 16-bit richiede l'assegnazione da parte di un PAN coordinator ma in TinyOS l'assegnazione dell'indirizzamento è manuale. Il metodo scelto per l'assegnazione dell'indirizzamento prevede l'utilizzo dalla parte finale del MAC address del chip radio, mediante la modifica del file contiki-sky-main.c nella funzione set_rime_addr() e nella funzione main().
Il metodo utilizzato dovrebbe evitare sovrapposizioni, finchè si utilizza hardware omogeneo per i test di interoperabilità.
Nel file uip-nd6.h sono stati modificati i paramentri delle opzioni dei messaggi di neighbor discovery, portandoli alla modalità short address.
Nel file uip-netif.c è stata modificata la procedura di autoconfigurazione dell'indirizzamento IID IPv6, rispettando lo standard per l'indirizzamento IEEE 802.15.4 a 16 bit con PAN ID.
In particolare, è da notare il bit 7 del byte più significativo dell'IID assegnato a 0, per indicare l'indirizzamento locale.
Inoltre, è stato modificato nel file uip-netif.c l'assegnazione dell'indirizzo di destinazione del messaggio ND NS corrisponde al proprio indirizzo solicited multicast nella funzione uip_netif_init().
Nel file framer-802154.c che contiene le funzioni per la creazione del frame, sono stati modificati i parametri del source e destination address IEEE 801.15.4 dalla modalità long alla modalità short.
Il test di funzionamento dell'indirizzamento 802.15.4 a 16-bit è stato realizzato tramite le applicazioni UPD-SENDER e UDP-RECEIVER che consentono di inviare da sender al receiver il messaggio "Sender says Hi!" e la risposta dal receiver al sender del messaggio "Receiver says Hi!".

### 5.6.2 Downgrade di 6lowpan

L'utilizzo da parte del sistema operativo TinyOS 2.x della versione draft-hui-6lowpan-hc-00 ha richiesto l'utilizzo della versione di Contiki più vicina a quella implementazione. In particolare, gli sviluppatori di Contiki hanno adottato come prima versione di 6lowpan la versione draft-hui-6lowpan-hc-01, nella versione Contiki 2.3.
Partendo dalla versione 2.3 sono state adeguate le funzioni di compression e decompression 6lowpan allineandole alle specifiche del draft ed adottando le stesse

semplifaccioni presenti in TinyOS 2.x, come ad esempio la scelta di non elidere l'hop count. In appendice C si riportano integralmente le funzioni modificate per la compressione e decompressione 6lowpan hc-00 del file sicslowpan.c e le variazioni apportate al file sicslowpan.h.

### 5.6.3 Adeguamento del protocollo ND

Il sistema operativo TinyOS 2.x non utilizza i messaggi Neighbor Solicitation (NS) e Neighbor Advertisement (NA) per la verifica di raggiungibilità dei nodi confinanti ma ricava l'indirizzo link-layer a partire dall'indirizzamento IPv6 ed invia direttamente il paccchetto. In Contiki 2.3, file uip-nd6.c sono state disattivate le funzioni periodiche di query del nodo confinante ed è stata adottata la stessa tecnica per ricavare l'indirizzo link-layer.

### 5.6.4 Associazione tra IPv6 local-link ed indirizzo link-layer

In TinyOS 2.x l'indirizzo IPv6 local-link si forma mantenendo a 0 i 48 bit più significati dell' IID, che rispetta la regola del 7° bit del byte più significativo dell'IID a 0. Per consentire l'interoperabilità è stata modificata la funzione di autoconfigurazione dell'indirizzamento IID nel file uip-netif.c.

### 5.6.5 Adeguamento header LOWPAN_UDP

Nel caso di compressione dell' UDP header, il draft-hui-6lowpan-hc-00 impone che il valore del primo bit (ID) del byte LOWPAN_NHC venga posto a zero. Dall'analisi delle frame si evidenzia che questo bit in TinyOS viene posto a 1, come nel caso non venga effettuata la compressione dell'header UDP. Per verificare l'interoperabilità del protocollo UDP, è stato modificato il primo bit, variando il valore da utilizzare nel caso di compressione dell'header UDP che si trova in sicslowpan.h.

## 5.7 Ambiente utilizzato per test di interoperabilità

L'ambiente utilizzato per i test di interoperabilità si componeva di un pc dotato di due immagini Linux virtualizzate tramite VMware, connesso tramite USB a 3 mote Crossbow Telos rev.b.
In particolare, è stato utilizzato un mote connesso con l'immagine virtuale contenente l'ambiente di sviluppo di Contiki 2.3, conosciuto come instant-contiki-2.3, nel quale sono state testate le varie applicazioni di Contiki 2.3, mentre sono stati utilizzati due mote connessi all'immagine virtuale contenente l'ambiente di sviluppo di TinyOS 2.1.
Il primo mote connesso all'ambiente di sviluppo TinyOS ha ospitato l'applicazione IPBasestation per la cattura dei pacchetti su rete 6lowpan mentre il secondo mote ha ospitato le varie applicazioni TinyOS necessarie per interagire con il mote Contiki.
I test sono stati effettuati disabilitando l'ack request a livello IEEE 802.15.4 in TinyOS Blip, tramite il parametro BLIP_L2_RETRIES.
Di seguito si riportano i serial-number dei mote utilizzati ed i ruoli assunti nei test:

| Serial number | Indirizzamento IEEE 802.15.4 | Ruolo |
|---|---|---|
| XBRDXCNP | 67:17 | *mote Contiki* |
| XBRDX7RB | 00:05 | *mote TinyOS* |
| XBRDXCHV | - | *IPBasestation per cattura pacchetti* |

Di seguito si riporta uno schema dell'ambiente utilizzato durante i test:

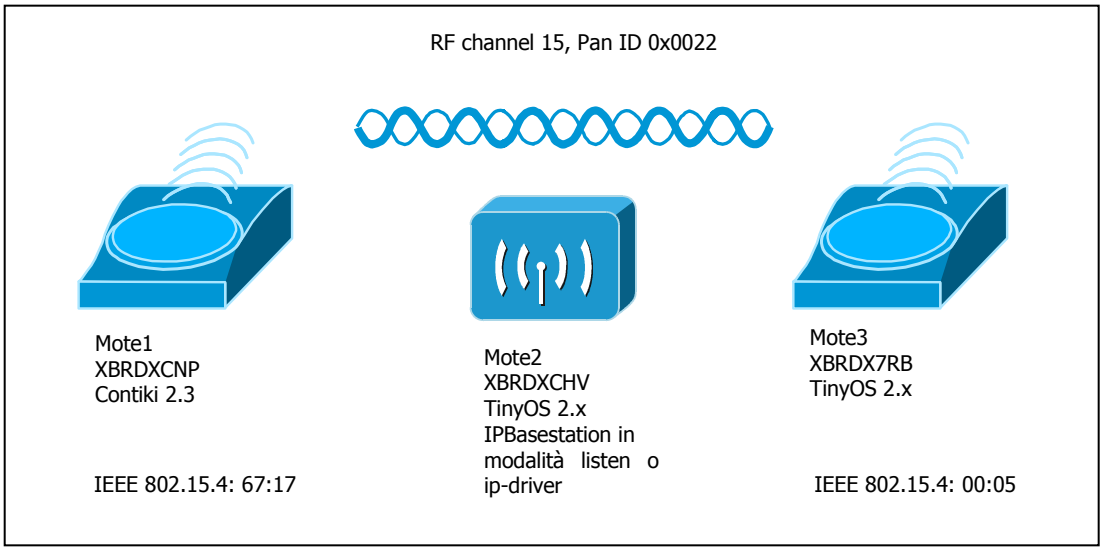


Figura 5.1 Interoperability lab

## 5.8 Test di interoperabilità

### 5.8.1 Level 0 test icmp echo ed echo-reply

Per verificare la risposta al ping da parte di Contiki è stata caricata nel mote 3 l'applicazione UDPHello, modificata per inviare icmp echo e nel mote 1 l'applicazione empty. Di seguito si riporta la cattura effettuata dal mote 2, nella quale si può evidenziare il byte 80, che evidenzia un pacchetto icmp echo ed il byte 81 che evidenzia un pacchetto icmp echo-reply.

```
02 1F 41 88 00 22 00 17 67 05 00 03 94 3A F0 00 05 67 17 80 00 17 8F 00 00 00 00 00 00 04 0C 1E EA
02 1F 41 88 3D 22 00 05 00 17 67 03 94 3A 40 67 17 00 05 81 00 16 8F 00 00 00 00 00 00 04 0C 10 EB
```

Per verificare la risposta al ping da parte di TinyOS, è stata caricato nel mote 3 l'applicazione UDPHello di TinyOS e nel mote 1 l'applicazione ping-ipv6 di Contiki 2.3. Di seguito si riporta la cattura effettuata dal mote 2, nella quale si può evidenziare il byte 80, che identifica un pacchetto icmp echo ed il byte 81 che identifica un pacchetto icmp echo-reply.

```
02 2B 41 88 3D 22 00 05 00 17 67 03 94 3A 40 67 17 00 05 80 00 1B 8F 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 10 EC
```

```
02 2B 41 88 6C 22 00 17 67 05 00 03 94 3A F0 00 05 67 17 81 00 1A 8F 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 1E EA
```

### 5.8.2 Level 1 test UDP con compressione

Per verificare la comunicazione UDP con compressione dell'header, è stata caricata nel mote 3 l'applicazione UDPHello di TinyOS e nel mote 1 l'applicazione udp-sender-ipv6 di Contiki 2.3. In entrambe le applicazioni è stata definita la porta di destinazione del pacchetto UDP verso TinyOS con il valore F0B0 e la porta di origine in Contiki F0B1.
Di seguito si riporta la cattura effettuata dal mote 2, nella quale si può evidenziare il byte di compressione dell'header UDP di valore E0 ed il byte seguente 10, che denota la porta di origine UDP compressa di valore 1 rispetto alla base F0B0 e di destinazione 0 rispetto alla base F0B0. Ovviamente il pacchetto di risposta vedrà i campi di compressione delle porte UDP invertite.

```
02 25 41 88 7F 22 00 05 00 17 67 03 D4 40 67 17 00 05 E0 10 34 8D 53 65 6E 64 65 72 20 73 61 79
73 20 48 69 21 15 E9
02 21 41 88 67 22 00 17 67 05 00 03 D4 F0 00 05 67 17 E0 01 F9 72 52 69 63 65 76 75 74 6F 20 21 00
1C EC
```

### 5.8.3 Level 2 test udp con compressione e frammentazione

Dopo aver corretto alcuni bug nelle funzioni di gestione del buffer 6lowpan in Contiki 2.3, è stata caricata nel mote 1 l'applicazione udp-sender-ipv6 di Contiki 2.3 ed nel mote 3 l'applicazione UDPHello di TinyOS. In entrambe le applicazioni è stata definita la porta di destinazione del pacchetto UDP verso TinyOS con il valore F0B0 e la porta di origine in Contiki F0B1.
Di seguito si riporta la cattura effettuata dal mote 2 composta da due frame, nella quale si può evidenziare il dispatch C0, che evidenzia il primo frame in caso di frammentazione ed il dispatch E0 che evidenzia il frame non iniziale in caso di frammentazione.

```
02 6A 41 88 51 22 00 05 00 17 67 C0 99 00 0A 03 D4 40 67 17 00 05 E0 10 41 48 51 75 65 73 74 6F
20 74 65 73 74 6F 20 73 69 63 75 72 61 6D 65 6E 74 65 20 67 65 6E 65 72 61 20 6C 61 20 66 72 61
6D 6D 65 6E 74 61 7A 69 6F 6E 65 20 65 20 63 69 20 70 65 72 6D 65 74 74 65 20 64 69 20 76 65 72
69 66 69 63 61 72 65 20 73 65 12 EB
02 29 41 88 52 22 00 05 00 17 67 E0 99 00 0A 10 20 73 69 20 72 69 65 73 63 65 20 61 20 72 69 63 6F
73 74 72 75 69 72 6C 6F 12 EB
02 21 41 88 48 22 00 17 67 05 00 03 D4 F0 00 05 67 17 E0 01 F9 72 52 69 63 65 76 75 74 6F 20 21 00
1E EC
```

### 5.8.4 Test icmp echo ed echo-reply con global prefix

Come ulteriore esplorazione degli ambiti di interoperabilità tra Contiki 2.3 e TinyOS 2.x, è stata testata la comunicazione con indirizzamento globale utilizzando l' edge router TinyOS in rete 6LoWPAN ed è stato verificato che tale comunicazione non può avvenire poichè Contiki non supporta funzioni di registrazione all'edge router, definite in draft-ietf-6lowpan-nd-0x, ma si appoggia all'RFC2461 IPv6 ND.

Il test utilizzato ha richiesto l'installazione dell'applicazione TinyOS IPBasestation in modalità ip-driver, cioè edge router, con indirizzamento aaaa::64 nel mote 3 e l'applicazione Contiki empty nel mote 1, con indirizzamento aaaa::6717.
Di seguito si riporta la cattura dell'interazioene tra nodo Contiki con un pacchetto RS e l'IPBasestation che risponde con un pacchetto RA, ma a seguire da parte di Contiki non viene generato alcun pacchetto di registratione all'edge router.

```
02 22 41 88 3C 22 00 FF FF 17 67 04 B4 3A 67 17 A4 02 85 00 AD FF 00 00 00 00 01 01 67 17 00 00
00 00 0F EA
02 4B 41 88 09 22 00 FF FF 64 00 03 94 3A FF 00 64 A4 01 86 00 74 1F 40 00 00 00 00 00 00 00 00
00 00 03 04 40 C0 00 27 8D 00 00 09 3A 80 00 00 00 00 AA AA 00 00 00 00 00 00 00 00 00 00 00
00 00 64 11 01 00 00 00 00 00 00 15 EB
02 46 41 88 3D 22 00 FF FF 17 67 04 A0 3A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 02
00 00 00 00 00 00 00 00 00 01 FF 00 67 17 87 00 01 CF 00 00 00 00 AA AA 00 00 00 00 00 00 00 00
00 00 00 00 67 17 06 EA
```

Di seguito invece si evidenzia il pacchetto di registrazione da parte del mote 1, nel quale è stata caricata l'applicazione TinyOS UDPHello, successivo al pacchetto RA da parte dell'edge router.

```
02 1B 41 88 00 22 00 FF FF 05 00 03 94 3A FF 00 05 A4 02 85 00 7D 32 00 00 00 00 12 EC
02 4B 41 88 0B 22 00 FF FF 64 00 03 94 3A FF 00 64 A4 01 86 00 74 1F 40 00 00 00 00 00 00 00 00
00 00 03 04 40 C0 00 27 8D 00 00 09 3A 80 00 00 00 00 AA AA 00 00 00 00 00 00 00 00 00 00 00
00 00 64 11 01 00 00 00 00 00 00 16 EB
02 1D 61 88 01 22 00 64 00 05 00 04 94 3C 41 00 05 AA 01 3B 0A 0A 08 0D 1C FF 00 00 64 12 EC
```

La fase di registrazione del nodo Contiki all'edge router può essere simulata con le seguenti operazioni: inserimento di righe di routing nel pc Linux connesso all'edge router e configurazione dell'edge router del nodo aggiuntivo.
La configurazione dell'edge router, si effettua tramite il comando add, utilizzando i valori decimali dell'indirizzo IEEE 802.15.4 dell'ip-driver e del nodo da aggiungere.
La verifica di raggiungibilità può esssere eseguita mediante un ping utilizzando l'interfaccia tun0 con destinazione l' indirizzamento globale.
Di seguito le catture delle comunicazioni registrate:

```
02 53 41 88 1A 22 00 17 67 64 00 04 94 3A 40 00 64 67 17 80 00 54 C4 89 23 00 16 57 C1 CD 4B CD
A6 07 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23
24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 1B E8
02 53 41 88 53 22 00 64 00 17 67 04 94 3A 40 67 17 00 64 81 00 53 C4 89 23 00 16 57 C1 CD 4B CD
A6 07 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23
24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 16 EC
```

La verifica di raggiungibilità può esssere eseguita mediante un ping utilizzando l'interfaccia tun0 con destinazione l' indirizzamento link-local.
Di seguito le catture delle comunicazioni registrate:

```
02 53 41 88 2B 22 00 17 67 64 00 03 94 3A 40 00 64 67 17 80 00 A8 E2 8E 23 00 0E D2 C1 CD 4B
4C E3 0C 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 1C E8
02 53 41 88 64 22 00 64 00 17 67 03 94 3A 40 67 17 00 64 81 00 A7 E2 8E 23 00 0E D2 C1 CD 4B
4C E3 0C 00 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 15 EB
```

# 6. Conclusioni

Questo lavoro ha permesso di dimostrare che le implementazioni Contiki 2.3 e TinyOS 2.x non utilizzano stack 6lowpan compatibili e possono dialogare solo mediante l'adeguamento dello stack di Contiki 2.3 allo stack di TinyOS 2.x, che è stato sviluppato sulla base di draft più datati.
Lo stack di TinyOS 2.x evidenzia alcune anomalie rispetto al draft draft-hui-6lowpan-hc-00 adottato ed ancora non si è adeguato all'evoluzione dei draft draft-ietf-6lowpan-hc-06 ed draft-ietf-6lowpan-nd-08.
Inoltre, TinyOS 2.x non supporta elementi fondamentali come l'indirizzamento IEEE 802.15.4 a 64 bit.
Lo stack Contiki è certificato IPv6 ready, implementa draft di compressione dell'header più recenti ma non implementa elementi fondamentali come 6lowpan ND secondo il draft draft-ietf-6lowpan-nd-08.
In questa fase dell'evoluzione della tecnologia alla base delle comunicazioni IPv6 dei Wireless Sensor Networks, nella quale il protocollo 6lowpan non è consolidato, si percepisce che lo sviluppo dei sistemi operativi si prefigge il raggiungimento della leadership, per imporre sul mercato la soluzione con la quale gli altri competitor dovranno garantire l'interoperabilità.

## Appendice A: Moduli di Contiki 2.3

Di seguito si riporta l'elenco dei moduli più importanti che costituiscono Contiki nella versione 2.3, con l'elenco dei file e le relative funzioni:

| Modulo | Descrizioni | files | funzioni |
|---|---|---|---|
| CTK VNC server | VNC server | ctk-vncserver.c | vnc_server_update_add, vnc_server_update_alloc, vnc_server_update_free, vnc_server_update_dequeue, vnc_server_update_remove, ctk_draw_init, ctk_draw_widget, ctk_draw_clear_window, ctk_draw_window, ctk_draw_dialog, ctk_draw_clear, ctk_draw_menus, ctk_draw_height, ctk_draw_width, ctk_arch_draw_char, ctk_arch_keyavail, ctk_arch_getkey |
| CTK graphical user interface | Il Contiki Toolkit (CTK) fornisce la GUI per il sistema Contiki | ctk-draw.h, ctk.c e ctk.h | ctk_mode_set, ctk_mode_get, ctk_window_new, ctk_window_clear, ctk_window_close, ctk_window_redraw, ctk_menu_add, ctk_menu_remove |
| CTK device driver function [CTK graphical user interface] | Diviso in due moduli, ctk-draw e ctk-arch, permettono di realizzare l'interfaccia tra CTK e l'hardware | ctk-conio.c, ctk-vncserver.c, ctk.h | ctk_draw_clear, ctk_draw_clear_window, ctk_draw_init, ctk_draw_menus, ctk_draw_widget |
| CTK events [CTK graphical user interface] | Eventi disponibili per CTK | ctk.c | |
| CTK application functions [CTK graphical user interface] | Funzioni CTK utilizzate dalle applicazioni | ctk.h | ctk_mode_set, ctk_mode_get, ctk_icon_add, ctk_window_open, ctk_window_close, ctk_window_clear, ctk_menu_add, ctk_menu_remove, ctk_window_redraw, ctk_window_new, ctk_menu_new, ctk_menuitem_add, ctk_widget_redraw, ctk_widget_add, ctk_desktop_width, ctk_desktop_height |
| Best-effort multihop forwarding | Implementa un meccanismo di multihop forwarding | rmh.c, rmh.h | |
| Single-hop reliable bulk data transfer | il Modulo rudolph2 implementa un meccanismo di trasferimento dati single-hop | rudolph2.c, rudolph2.h | |
| Function quick reference | Alcuni sensori generano eventi al cambiamento dei sensori ed è possibile verificare lo stato dei sensori attraverso funzioni di query | | sensors_battery, sensors_button, sensors_mic, sensors_pir, sensors_radiosignal, sensors_temp, sensors_vib, leds_on, leds_off, leds_invert, leds_blink, beep, beep_beep, beep_down, beep_quick, beep_spinup, etimer_expired, etimer_set, etimer_reset, etimer_restart, timer_expired, timer_set, timer_reset, timer_restart |
| **Communication stacks** | | | |
| uIP TCP/IP stack | uIP TCP/IP stack | uip.c, uip.h | uip_setipid, uip_add32, uip_chksum, uip_ipchksum, uip_tcpchksum, uip_init, uip_udp_new, uip_unlisten, uip_listen, uip_process. htons, uip_send, uip_udpchksum, uip_icmp6chksum |
| **The uIP TCP/IP stack** | | | |
| Protosockets library | libreria di interfaccia con l'uIP stack, solo per TCP, utilizza Protothreads | psock.h | |
| uIP hostname resolver functions | Le funzioni uIP DNS resolver sono utilizzate per risolvere un nome host ad un IP | resolv.c | resolv_query, resolv_lookup, resolv_getserver, resolv_conf |
| Serial Line IP (SLIP) protocol | SLIP permette l'invio di pacchetti IP su linea seriale | slipdev.c, slipdev.h | slipdev_send, slipdev_poll, slipdev_init, slipsev_char_put, slipdev_char_poll |
| The Contiki/uIP interface | Contiki fornisce funzioni per la gestione delle | tcpip.h | TCP/IP packet processing: tcpip_do_forwarding, tcpip_is_forwarding, tcpip_input, tcpip_output, |

| | | | |
|---|---|---|---|
| | connessioni, per garantire che le connessioni siano legate al processo corretto | | tcpip_set_outputfunc, tcpip_set_forwarding<br><br>TCP functions:<br>tcp_attach, tcp_listen, tcp_unlisten, tcp_connect, tcpip_poll_tcp, tcp_markconn<br><br>UDP functions:<br>udp_attach, udp_new, udp_broadcast_new, tcpip_poll_udp |
| uIP packet forwarding | | uip-fw.c, uip-fw.h | uip_fw_init, uip_fw_output, uip_fw_forward, uip_fw_register, uip_fw_default, uip_fw_periodic |
| uIP TCP throughput booster hack | L'implementazione uIP TCP permette ad ogni connessione TCP di avere un solo segmento in uscita contemporaneamente. Il modulo uip-split module è un tentative di rimediare a questo limite, splittando in 2 la massima dimensione del segmento TCP in uscita ed utilizza uip-fw module per l'invio dei pacchetti. | uip-split.h | uip_split_output |
| uIP initialization functions | Le funzioni di inizializzazione uIP servono al boot di uIP | uip.c | uip_init, uip_setipid |
| uIP device driver functions | Funzioni utilizzate da un network device driver per interagire con uIP | uip.h | uip_reass_over |
| uIP application functions | funzioni utilizzate dalle applicazioni sopra uIP | uip.h | uip_listen, uip_unlisten, uip_connect, uip_send, uip_udp_new |
| uIP conversion functions | funzioni da utilizzare per convertire i diversi formati utilizzati da uIP | uip.h | htons, uiplib_ipaddrconv |
| uIP Address Resolution Protocol | Address Resolution Protocol (supporto solo per Ehernet) | uip_arp.c, uip_arp.h | uip_arp_init, uip_arp_timer, uip_arp_arpin, uip_arp_out |
| Configuration options for uIP | uIP è configurato utilizzando una configurazione "di progetto" | uipopt.h, uip-conf.h | |
| 6LoWPAN implementation | implementazione di 6LoWPAN, solo su indirizzi 64-bit | sicslowpan.h, sicslowpan.c | sicslowpan_init |
| uIP IPv6 specific features | lo stack uIP IPv6 fornisce il supporto per la comunicazione a Contiki | uip-icmp6.c, uip-icmp6.h, uip-nd6-io.c, uip-nd6.c, uip-nd6.h, uip-netif.c, uip-netif.h, uip6.c, rimeroute.c, rimeroute.h | **icmp:**<br>uip_icmp6_echo_request_input, uip_icmp6_error_output<br>**ND neighbor cache, router list e prefix handling:**<br>uip_nd6_init, uip_nd6_nbrcache_lookup, uip_nd6_nbrcach_add, uip_nd6_defrouter_lookup, uip_nd6_choose_defrouter, uip_nd6_defrouter_rm, uip_nd6_defrouter_add, uip_nd6_is_addr_onlink, uip_nd6_prefix_lookup, uip_nd6_prefix_add, uip_nd6_prefix_rm, uip_nd6_periodic<br>**ND Messages Processing and Generation:**<br>uip_nd6_io_ns_input, uip_nd6_io_ns_output, uip_nd6_io_na_input, uip_nd6_io_rs_output, uip_nd6_io_ra_input<br>**ICMPv6 variables:**<br>uip_chksum, uip_ipchksum, uip_icmp6chksum, uip_init, uip_udp_new, uip_process, htons, htonl, uip_send<br>Other Functions:<br>uip_netif_init, uip_netif_periodic, uip_netif_compute_reachable_time, uip_netif_is_addr_my_solicited, uip_netif_addr_lookup, uip_netif_addr_add, uip_netif_addr_autoconf_set, uip_netif_select_src, uip_netif_sched_dad, uip_netif_dad, uip_netif_dad_failed, |

| | | | |
|---|---|---|---|
| | | | uip_netif_sched_send_rs, uip_netif_send_rs |
| Architecture specific uIP functions | La calcolazione dell' IP checksum è l'operazione più costosa in TCP/IP e per questo deve essere implementata in modo ottimizzato in base all'architettura | uip_arch.h | uip_addr32, uip_chksum, uip_ipchksum, uip_tcpchksum |
| **Configuration options for uIP** | | | |
| Static configuration options | Options valide solo per IPv4 e possono essere utilizzate solo se UIP_FIXEDADDRES==1 | uipopt.h | |
| IP configuration options | Defines per IP TTL, maximum time for reassembly e attivazione supporto per IP packet reassembly | uipopt.h | |
| IPv6 configuration options | Defines per MTU, IPv6, neighboor queuing, IPv6 consistency check, IPv6 fragmentation, indirizzi e prefissi IPv6 associati con l'interfaccia del nodo, numero di neighbors da registrare in cache, numero minimo di default router | uipopt.h | |
| UDP configuration options | Defnines per opzioni per il supporto UDP (in uIP supporto UDP non è completo per multicast e broadcast), per indicare se inserire in compilazione il supporto UDP e se utilizzare UDP checksum, numero max di UDP contemporanee | uipopt.h | |
| TCP configuration options | Defines per opzioni del supporto TCP, se TCP deve essere inserito in compilazione, numero max di connessioni TCP, numero max porte TCP in ascolto, se inserire in compilazione il supporto per urgent data notification, TCP MSS, dimensione del receiver window, quanto tempo una connessione deve rimanere in TIME_WAIT | uipopt.h | |
| ARP configuration options | Defnies per configurazione dimensione ARP table e max age time | uipopt.h | |
| layer 2 options (for ipv6) | | | |
| 6lowpan options (for ipv6) | defines per configurazione parametric 6lowpan, frammentazione, 6lowpan timeout per riassemblaggio pacchetti a livello 6lowpan, compressione header, utilizzo compressione IPHC | | |
| General configuration options | Defines e funzioni per la configurazione delle dimensioni del buffer, logging, supporto broadcast, lunghezza header a livello link | uipopt.h | uip_log |
| CPU architecture configuration | Define per specificare l'endianess della CPU, cioè l'ordine di registrazione e lettura dei byte in memoria (little o | uipopt.h | |

| | | | |
|---|---|---|---|
| | big) | | |
| Application spefic configurations | In merito al nome dell'applicazione da chiamare in caso di evento TCP/IP | uipopt.h | |
| **Communication stacks** | | | |
| The Rime communication stack | Stack che fornisce un insieme di primitive di comunicazione lightweight. | rime.h, rime.c | rime_init, rime_input, rime_driver_send |
| **The Rime communication stack** | | | |
| Anonymous best-effort local area broadcast | Il modulo abc invia pacchetti a tutti i nodi confinanti locali. | abc.c, abc.h | abc_open, abc_close, abc_send, abc_input |
| Announcements | Le primitive announcement inviano announcement sulla lan | announcement.c, announcement.h | **System API:** announcement_init, announcement_register_listen_callback, announcement_register_observer_callback, announcement_list, announcement_heard **Application API:** announcement_register, announcement_remove, announcement_set_value, announcement_set_id, announcement_listen |
| Best-effort local area broadcast | Il modulo broadcast invia pacchetti a tutti i confinanti, con header che identifica il mittente | broadcast.h | broadcast_open, broadcast_close, broadcast_send |
| Tree-based hop-by-hop reliable data collecton | Il modulo realizza un meccanismo di hop-by-hop data collection affidabile | collect.c, collect.h | |
| Callback timer | Meccanismo per chiamare un funzione alla scadenza del timer | ctimer.c, ctimer.h | |
| Ipolite best effort local broadcast | Questo modulo invia un pacchetto broadcast ad ogni intervallo temporale | ipolite.c, ipolite.h | ipolite_open, ipolite_close, ipolite_send, ipolite_cancel |
| Mesh routing | Il modulo mesh invia pacchetti utilizzando routing multi-hop ad un destinatario specifico | mesh.c, mesh.h | mesh_open, mesh_close, mesh_send |
| Best-effort multihop forwarding | Questo modulo implementa multihop forwarding, il routing deve essere già settato, con uno dei moduli Rime. | multihop.h | |
| Neighbor discovery | Questo modulo implementa un meccanismo di discovery periodica dei nodi confinanti | neighbor-discovery.c, neighbor-discovery.h | |
| Rime neighbor management | Questo modulo gestisce la neighbor table | neighbor.c, neighbor.h | |
| Best-effort network flooding | Questo modulo effettua best-effort flooding, inviando un pacchetto a tutti i nodi, mediante polite broadcast ad ogni hop | netfllod.c, netflood.h | |
| Rime buffer management | Effettua la gestione del buffer di Rime | packetbuf.c, packetbuf.h | packetbuf_clear, packetbuf_copyfrom, packetbuf_compact, packetbuf_copyto_hdr, packetbuf_copyto, packetbuf_hdralloc, packetbuf_hdrreduce, packetbuf_set_datalen, packetbuf_dataptr, packetbuf_hdrptr, packetbuf_reference, packetbuf_is_reference, packetbuf_reference_ptr, packetbuf_datalen, packetbuf_hdrlen, packetbuf_totlen |
| Packet queue | gestisce la lista dei pacchetti in coda | packetqueue.c, packetqueue.h | **Packet queue functions:** packetqueue_init, packetqueue_enqueue_packetbuf, packetqueue_first, packetqueue_dequeue **Packet queue item functions:** packetqueue_queuebuf, packetqueue_ptr |
| Rimepoliteannouncement | Effettua un annuncio periodico, annunci | polite-announcement.h | |

| | | | |
|---|---|---|---|
| | registrati con l'announcement module | | |
| Polite anonymous best effort local broadcast | Invia broadcast nella local area ogni intervallo temporale | polite.c, polite.h | polite_open, polite_close, polite_send, polite_cancel |
| Rime queue buffer management | gestisce buffer accodati | queuebuf.c, queuebuf.h | |
| Rime addresses | Rappresentazione astratta degli indirizzi in Rime | rimeaddr.c, rimeaddr.h | rimeaddr_copy, rimeaddr_cmp, rimeaddr_set_node_addr |
| Rime route discovery protocol | In Rime effettua il discovery delle route | route-discovery.c, route-discovery.h | |
| Rime route table | In Rime gestisce la tabella di routing | route.c, route.h | |
| Single-hop reliable bulk data transfer | Il modulo rudolph implementa un single-hop bulk data transfer reliable | rudolph0.c, rudolph0.h | |
| Multi-hop reliable bulk data transfer | Il modulo rudolph1 implementa un multi-hop bulk data transfer reliable | rudolph1.c, rudolph1.h | |
| Single-hop reliable unicast | Questo modulo ad un solo single-hop nodo confinante | runicast.c, runicast.h | |
| Stubborn best-effort local area broadcast | Fornisce best-effort local area broadcast anonymous | stbroadcast.c, stbroadcast.h | stbroadcast_open, stbroadcast_set_timer, stbroadcast_send_stubborn, stbroadcast_cancel |
| Stubborn unicast | Invia ripetutamente un pacchetto ad un solo nodo confinante utilizzando la primitive unicast | stunicast.c, stunicast.h | |
| Reliable single-source multi-hop flooding | Invia pacchetti singoli a tutti i nodi del network | trickle.c, trickle.h | |
| Single-hop unicast | Invia un pacchetto ad un nodo confinante single-hop identificato | unicast.c, unicast.h | |
| **Memory functions** | | | |
| Memory block management functions | Gruppo di funzioni semplici e potenti per mappare blocchi di memoria di dimensioni fisse. | memb.c, memb.h | memb_init, memb_alloc, memb_free |
| Managed memory allocator | Mantiene la memoria libera da frammentazione, compattando la memoria quando si presentano blocchi liberi | mmem.c, mmem.h | mmem_alloc, mmem_free, mmem_init |
| **Contiki system** | | | |
| Implicit network time synchronization | modulo per il network time sync per tutti i nodi nel network | timesynch.c, timesynch.h | timesync_init, timesynch_time, timesynch_time_to_rtimer, timesynch_rtimer_to_time, timesynch_offset, timesynch_authority_level, timesynch_set_authority_level |
| The Contiki file system interface | Defnisce una API per la lettura delle directory e lettura e scrittura dei files | cfs-coffee.h, cfs.h | Per programmi applicativi: cfs_coffee_reserve, cfs_coffee_configure_log, cfs_coffee_format, cfs_coffee_get_protected_mem Functions: cfs_open, cfs_close, cfs_read, cfs_write, cfs_seek, cfs_remove, cfs_opendir, cfs_readdir, cfs_closedir |
| Argument buffer | Può essere utilizzato per il passaggio da un processo in fase di conclusione ad un processo che ancora non è stato creato | arg.c | arg_alloc, arg_free |
| Clock library | Interfaccia tra il Contiki ed le funzionalità specifiche del clock della della piattaforma | clock.c | clock_init, clock_time, clock_delay |
| Communication power accounting | Questo modulo raccoglie le informazioni sul consumo di energia in relazione alle attività di comunicazione | compower.c, compower.h | compower_init, compower_accumulate, compower_clear, compower_attrconv, compower_accumulate_attrs |

| | | | |
|---|---|---|---|
| Event timers | Questo modulo fornisce un modo per generare eventi temporizzati. | etimer.c, etimer.h | **Called by the system from timer interrupts:** etimer_request_poll, etimer_pending, etimer_next_expiration_time<br>Called from application programs: etimer_set, etimer_reset, etimer_restart, etimer_adjust, etimer_expired, etimer_expiration_time, etimer_start_time, etimer_stop |
| The Contiki program loader | Interfaccia per il caricamento e lancio di programmi | loader.h | |
| Contiki process | Un processo in Contiki è constituito da un singolo protothread | process.c, process.h | **Chiamate dai programmi applicativi:** process_current, process_alloc_event, process_start, process_exit, process_post, process_post_synch, process_current, process_context_begin, process_context_end<br>**Chiamate dal system e dal codice di boot-up:** process_init, process_run, process_nevents, process_is_running<br>**Chiamate dai driver del device:** process_poll |
| Real-time task scheduling | Questo modulo gestisce la schedulazione l'esecuzione dei task real-time, con tempi d'esecuzione prevedibili | rtimer.c, rtimer.h | rtimer_init, rtimer_set, rtimer_run_next |
| Seconds timer library | Fornisce le funzioni per impostare, reimpostare e per verificare se un timer è scaduto | stimer.c, stimer.h | stimer_set, stimer_reset, stimer_restart, stimer_expired, stimer_remaining |
| Contiki subprocess | Subprocess è un processo all'interno di un processo | subprocess.h | |
| **The Contiki ELF loader** | | | |
| The Contiki ELF loader | Carica oggetti ELF (Executable linkable format) in un sistema Contiki attivo | elfload.h, elfloader-otf.h | elfloader_init, elfloader_load, elfloader_allocate_segment, elfloader_start_segment, elfloader_end_segment, elfloader_write_segment, elfloader_segment_offset, elfloader_load |
| Architecture specific functionality for the ELF loader | Questa funzionalità deve essere implementata per ogni tipo di processore sul quale Contiki funziona | elfloader-arch.h, elfloader-arch-otf.h | elfloader_arch_allocate_ram, elfloader_arch_allocate_rom, elfloader_arch_relocate, elfloader_arch_write_rom |
| **Multi-threading library** | | | |
| Multi-threading library | Preemptive multi-threading è implementato come library che può essere linkata opzionalmente con l'applicazione | mt.c | mt_init, mt_remove, mt_start, mt_exec, mt_yield, mt_exit, mt_stop |
| Architecture support for multi-threading | La libreria di Contiki per il multi-threading richiede il supporto specifico da parte dell'architettura per il set-up e lo switching degli stack | mt.h | mtarch_init, mtarch_remove, mtarch_start, mtarch_exec, mtarch_yield, mtarch_stop |
| **Timer library** | | | |
| Timer library | Il kernel non fornisce supporto per gli eventi temporizzati e questa libreria serve per le applicazioni che richiedono questo tipo di eventi | timer.c, timer.h | timer-set, timer_reset, timer_restart, timer_expired, timer_remaining |
| **Protothreads, tipo di thread senza stack che fornisce l'esecuzione del codice per sistemi event-driven** | | | |
| Local continuation | La base per l'implementazione del protothreads. | pt.h, lc-addrlabels.h, lc-switch.h | |
| Protothread semaphores | Implementa primitive di sincronizzazione (semaphores), "wait" e "signal" | pt-sem.h | |
| **Device driver APIs** | | | |
| EEPROM API | interfaccia per l'accesso alla EEPROM da Contiki | eeprom.h | eeprom_write, eeprom_read, eeprom_init |

| | | | |
|---|---|---|---|
| LEDs API | funzioni per l'access ai LEDs nelle piattaforme dotate di LEDs | leds.c | leds_blink, leds_get, leds_arch_init |
| Radio API | funzioni che devono essere implementate dai driver dalla radio | radio.h | |
| **The Tmote Sky Board** | | | |
| The Tmote Sky Board | Platform-specific source code | | in directory platfrom/sky e cpu/msp430<br>Per la scrittura inflash ROM: cpu/msp430/flsh.c<br>Per la lettura e scrittura da flash esterna: platform/sky/dev/xmem.c<br>Per l'accesso a serial/USB: cpu/msp430/dev/uart1.c oppure platfrom/sky/slip_uart1<br>Driver CC2420: core/dev/simple-cc2420.c e core/dev/cc2420.c |
| **Libraries** | | | |
| CRC16 calculation | Hash function per identificare errori di trasmissione | crc16.c, crc16.h | crc16_add, crc16_data |
| Linked list library | Funzioni per la gestione delle liste concatenate | list.c, list.h | list_init, list_head, list_copy, list_tail, list_add, list_push, list_chop, list_pop, list_remove, list_length, list_insert |
| Table-driven Manchester encoding and decoding | Funzioni per la codifica Manchester | me.c, me.h | me_encode, me_decode16, me_decode8, me_valid |
| Ring buffer library | Implementazione di un ring buffer | ringbuf.h | ringbuf_init, ringbuf_put, ringbuf_get, ringbuf_size, ringbuf_elements |
| | | | |
| Rimebroadcast | identifica broadcast best-effort in local area | broadcast.c | broadcast_open, broadcast_close, broadcast_send |
| Rimemh | Multihop forwarding | multihop.c | |
| Frame802154 | Gestione delle frame 802.15.4 | frame802154.c, frame802154.h | frame802154_hdrlen, frame802154_create, frame802154_parse |
| XgSmscRegs | | lanc111.c | |
| XgNicLanc111 | | | |
| Usb | | config.h | |
| PLL Macros [Usb] | Per il controllo di PLL | | |
| **Wireless, radio driver per chip Atmel AT86RF230, 231 e 212** | | | |
| RF230 MAC | Primitive mac per IEEE 802.15.4 | mac.c, zmac.h, ieee-15-4-manager.c, ieee-15-4-manager.h | mac_init, ieee_15_4_init |
| SICSLoWMAC Imlementation | Implementazione fase 1 MAC per supportare IPv6/6LoWPAN stack | mac.h, sicslowpan.c, | |
| RF230 Frame handling | creazione e parsing frame 802.15.4 | frame.c, frame.h | frame_tx_create, rx_frame_parse |
| RF230 hardware level drivers | low-level radio driver | hal.c, hal.h | hal_init, hal_reset_flags, hal_get_bat_low_flag, hal_clear_bat_low_flag, hal_get_trx_end_event_handler, hal_set_trx_end_event_handler, hal_clear_trx_end_ebent_handler, hal_get_rx_start_event_handler, hal_set_rx_start_event_handler, hal_get_pll_lock_flag, hal_clear_pll_lock_flag, hal_register_read, hal_register_write, hal_subregister_read, hal subregister_write, hal_frame_read, hal_frame_write, hal_sram_read, hal_sram_write, RADIO_VECT, TIMER1_OVF_vect |
| RF230 interface | radio driver code | radio.c, radio.h | radio_is_sleeping, radio_init, radio_get_operating_channel, radio_set_operating_channel, radio_get_tx_power_level, radio_set_tx_power_level, radio_get_cca_mode, radio_get_ed_threshold, radio_status_t radio_set_cca_mode, radio_status_t radio_get_rssi_value, radio_batmon_get_voltage_threshol, radio_batmon_get_voltage_range, radio_batmon_configure, radio_batmon_get_status, radio_get_clock_speed, radio_set_clock_speed, radio_calibrate_filter, radio_calibrate_pll, |

| | | | |
|---|---|---|---|
| | | | radio_get_trx_state<br>radio_set_trx_state, radio_enter_sleep_mode, radio_leave_sleep_mode,<br>radio_reset_state_machine, radio_reset_trx, radio_use_auto_tx_crc,<br>radio_send_data, radio_get_device_role, radio_set_device_role, radio_get_pan_id, radio_set_pan_id, radio_get_short_address, radio_set_short_address,<br>radio_get_extended_address,<br>radio_set_extended_address,<br>radio_configure_csma, calibrate_rc_osc_clkm, calibrate_rc_osc_32k |
| **Interfaces** | | | |
| UART1 | Gestione dei protocolli tramite UART1 | sdspi.c | uart_init, sdspi_init, sdspi_select, sdspi_rx, sdspi_tx, sdspi_read, sdspi_write, sdspi_idle, sdipi_wait_token |
| Libsd | | sd.h | sd_erase_blocks, sd_read_cid, sd_get_size |

## Appendice B: Dimensione dei moduli Contiki 2.3

Si riportano di seguito i dati di estensione delle singole componenti nella ROM e nella RAM del device con i dati comulativi delle estensioni dei moduli del sistema operativo.

| NOME OBJECT | DESCRIZIONE MODULO | COMPONENTE | TEXT length | DATA length | BSS length | COMMON |
|---|---|---|---|---|---|---|
| list.o | Linked list library [Libraries] | library: contiki (core/lib) | 234 | 0 | 0 | 0 |
| memb.o | Memory block management functions [Memory functions] | library: contiki (core/lib) | 320 | 0 | 0 | 0 |
| memchr.o | language library libc | library: contiki (core/lib) | 36 | 0 | 0 | 0 |
| memcmp.o | language library libc | library: contiki (core/lib) | 48 | 0 | 0 | 0 |
| memcpy.o | language library libc | library: contiki (core/lib) | 212 | 0 | 0 | 0 |
| memmove.o | language library libc | library: contiki (core/lib) | 212 | 0 | 0 | 0 |
| memset.o | language library libc | library: contiki (core/lib) | 112 | 0 | 0 | 0 |
| random.o | | library: contiki (core/lib) | 12 | 0 | 0 | 0 |
| ringbuf.o | Ring buffer library [Libraries] | library: contiki (core/lib) | 164 | 0 | 0 | 0 |
| | **TOTALE COMPONENTE** | **library: contiki** | **1350** | **0** | **0** | **0** |
| __stop_progExec__.o | low level language library libgcc | library: external (compiler-provided library functions) | 4 | 0 | 0 | 0 |
| _addsub_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 702 | 0 | 0 | 0 |
| _divmodhi4.o | low level language library libgcc | library: external (compiler-provided library functions) | 54 | 0 | 0 | 0 |
| _fixunssfsi.o | low level language library libgcc | library: external (compiler-provided library functions) | 62 | 0 | 0 | 0 |
| _fpcmp_parts_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 280 | 0 | 0 | 0 |
| _ge_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 98 | 0 | 0 | 0 |
| _mul_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 618 | 0 | 0 | 0 |
| _mulsi3hw.o | low level language library libgcc | library: external (compiler-provided library functions) | 40 | 0 | 0 | 0 |
| _pack_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 432 | 0 | 0 | 0 |
| _reset_vector__.o | low level language library libgcc | library: external (compiler-provided library functions) | 58 | 0 | 0 | 0 |
| _sf_to_si.o | low level language library libgcc | library: external (compiler-provided library functions) | 156 | 0 | 0 | 0 |
| _si_to_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 146 | 0 | 0 | 0 |
| _thenan_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 8 | 0 | 0 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| _udivmodhi4.o | low level language library libgcc | library: external (compiler-provided library functions) | 28 | 0 | 0 | 0 |
| _udivmodsi4.o | low level language library libgcc | library: external (compiler-provided library functions) | 42 | 0 | 0 | 0 |
| _unpack_sf.o | low level language library libgcc | library: external (compiler-provided library functions) | 246 | 0 | 0 | 0 |
| printf.o | language library libc | library: external (compiler-provided library functions) | 18 | 0 | 0 | 0 |
| puts.o | language library libc | library: external (compiler-provided library functions) | 46 | 0 | 0 | 0 |
| rand.o | language library libc | library: external (compiler-provided library functions) | 108 | 4 | 0 | 0 |
| snprintf.o | language library libc | library: external (compiler-provided library functions) | 42 | 0 | 0 | 0 |
| strncmp.o | language library libc | library: external (compiler-provided library functions) | 48 | 0 | 0 | 0 |
| vsnprintf.o | language library libc | library: external (compiler-provided library functions) | 72 | 0 | 4 | 0 |
| vuprintf.o | language library libc | library: external (compiler-provided library functions) | 1572 | 2 | 2 | 0 |
| | **TOTALE COMPONENTE** | **library: external** | **4880** | **6** | **6** | **0** |
| sicslowpan.o | SICSLoWMAC Imlementation [RF230 MAC] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 3008 | 0 | 34 | 0 |
| tcpip.o | The Contiki/uIP interface | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 1548 | 10 | 50 | 1 |
| uip6.o | uIP IPv6 specific features [The uIP TCP/IP stack] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 5210 | 3 | 12 | 790 |
| uip-icmp6.o | uIP IPv6 specific features [The uIP TCP/IP stack] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 540 | 0 | 16 | 0 |
| uip-nd6.o | uIP IPv6 specific features [The uIP TCP/IP stack] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 1968 | 0 | 1332 | 8 |
| uip-nd6-io.o | uIP IPv6 specific features [The uIP TCP/IP stack] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 3316 | 0 | 22 | 0 |
| uip-netif.o | uIP IPv6 specific features [The uIP TCP/IP stack] | networking: ipv6 (core/net/*.c assuming UIP_CONF_IPV6=1) | 1568 | 0 | 6 | 150 |
| | **TOTALE COMPONENTE** | **networking: IPv6** | **17158** | **13** | **1472** | **949** |
| frame802154.o | Frame802154 | networking: mac (core/net/mac) | 976 | 0 | 0 | 0 |
| framer.o | | networking: mac (core/net/mac) | 20 | 0 | 0 | 2 |
| framer-802154.o | Frame802154 | networking: mac (core/net/mac) | 416 | 4 | 1 | 0 |
| framer-nullmac.o | | networking: mac (core/net/mac) | 112 | 0 | 0 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| xmac.o | A simple power saving MAC protocol based on X-MAC | networking: mac (core/net/mac) | 2310 | 26 | 192 | 0 |
| | **TOTALE COMPONENTE** | **networking: mac** | **3834** | **30** | **193** | **2** |
| announcement.o | Announcements [The Time communication stack] | networking: rime (core/net/rime) | 224 | 4 | 4 | 0 |
| ctimer.o | Callback timer [The Rime communication stack] | networking: rime (core/net/rime) | 520 | 14 | 1 | 0 |
| packetbuf.o | Rime buffer management [The Rime communication stack] | networking: rime (core/net/rime) | 762 | 2 | 186 | 72 |
| queuebuf.o | Rime queue buffer management [The Rime communication stack] | networking: rime (core/net/rime) | 428 | 16 | 4126 | 3 |
| rimeaddr.o | Rime addresses [The Rime communication stack] | networking: rime (core/net/rime) | 88 | 0 | 0 | 8 |
| rimestats.o | | networking: rime (core/net/rime) | 0 | 0 | 0 | 72 |
| | **TOTALE COMPONENTE** | **networking: rime** | **2022** | **36** | **4317** | **155** |
| cc2420-arch.o | The Tmote Sky Board [Contiki Platform] | platform (platform/msp430) | 88 | 0 | 0 | 0 |
| crt430x1611.o | | platform (platform/msp430) | 6 | 0 | 0 | 0 |
| irq.o | | platform (platform/msp430) | 514 | 0 | 1 | 0 |
| leds-arch.o | LEDs API [Device driver APIs] | platform (platform/msp430) | 112 | 0 | 0 | 0 |
| light.o | The Tmote Sky Board [Contiki Platform] | platform (platform/msp430) | 64 | 0 | 0 | 0 |
| msp430.o | | platform (platform/msp430) | 294 | 2 | 0 | 0 |
| node-id.o | The Tmote Sky Board [Contiki Platform] | platform (platform/msp430) | 118 | 2 | 0 | 0 |
| rtimer-arch.o | Real-time task scheduling [Contiki system] | platform (platform/msp430) | 54 | 0 | 0 | 0 |
| uart1.o | The Tmote Sky Board [Contiki Platform] | platform (platform/msp430) | 422 | 0 | 74 | 0 |
| uart1-putchar.o | | platform (platform/msp430) | 14 | 0 | 0 | 0 |
| | **TOTALE COMPONENTE** | **platform** | **1686** | **4** | **75** | **0** |
| autostart.o | Implementation of module for automatically starting and exiting a list of processes [Contiki system] | system: base (core/sys) | 94 | 0 | 0 | 0 |
| clock.o | Clock library [Contiki system] | system: base (core/sys) | 316 | 4 | 4 | 0 |
| compower.o | Communication power accounting [Contiki system] | system: base (core/sys) | 150 | 0 | 8 | 8 |

| | | | | | | |
|---|---|---|---|---|---|---|
| contiki-sky-main.o | The Tmote Sky Board [Contiki Platform] | system: base (core/sys) | 1080 | 8 | 4 | 6 |
| empty.co | | system: base (core/sys) | 34 | 10 | 0 | 0 |
| energest.o | Communication power accounting [Contiki system] | system: base (core/sys) | 192 | 0 | 0 | 86 |
| etimer.o | Event timers [Contiki system] | system: base (core/sys) | 596 | 10 | 4 | 0 |
| process.o | Contiki process, the kernel [Contiki system] | system: base (core/sys) | 734 | 4 | 62 | 1 |
| rtimer.o | Real-time task scheduling [Contiki system] | system: base (core/sys) | 104 | 0 | 20 | 0 |
| sensors.o | | system: base (core/sys) | 586 | 10 | 6 | 1 |
| stimer.o | Seconds timer library [Contiki system] | system: base (core/sys) | 150 | 0 | 0 | 0 |
| timer.o | Timer library | system: base (core/sys) | 94 | 0 | 0 | 0 |
| | **TOTALE COMPONENTE** | **system: base** | **4130** | **46** | **108** | **102** |
| battery-sensor.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 96 | 0 | 0 | 0 |
| button-sensor.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 206 | 0 | 4 | 0 |
| cc2420.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 1968 | 10 | 12 | 10 |
| ds2411.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 444 | 0 | 0 | 8 |
| leds.o | LEDs API [Device driver APIs] | system: device drivers (core/dev) | 370 | 0 | 2 | 0 |
| sht11.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 682 | 0 | 0 | 0 |
| spi.o | | system: device drivers (core/dev) | 54 | 2 | 0 | 0 |
| watchdog.o | Monitor and signals if a thread is blocked | system: device drivers (core/dev) | 64 | 2 | 0 | 0 |
| xmem.o | The Tmote Sky Board [Contiki Platform] | system: device drivers (core/dev) | 872 | 0 | 0 | 0 |
| | **TOTALE COMPONENTE** | **system: device drivers** | **4756** | **14** | **18** | **18** |
| | | **TOTALE** | **39816** | **149** | **6189** | **1226** |
| | | **FILLER e VECTORS** | **32** | **1** | **3** | **4** |

**Appendice C: Codice delle funzioni 6lowpan Contiki modificate per HC-00**

Di seguito viene riportato il codice modificato per la funzione di compressione in sicslowpan.c:

```
static void
compress_hdr_hc00(rimeaddr_t *rime_destaddr)
{
  hc00_ptr = rime_ptr + 2;
  /*
   * As we copy some bit-length fields, in the IPHC encoding bytes,
   * we sometimes use |=
   * If the field is 0, and the current bit value in memory is 1,
   * this does not work. We therefore reset the IPHC encoding here
   */
  memset(RIME_IPHC_BUF->encoding, 0, 1);
  // verifica del context di apparteneza sulla base del destination address
  context = addr_context_lookup_by_prefix(&UIP_IP_BUF->destipaddr);
  if((context->number==0)||(uip_is_addr_mcast(&UIP_IP_BUF->destipaddr))){
  //se destination address local-link o multicast dispatch 0x03
  RIME_IPHC_BUF->dispatch = SICSLOWPAN_DISPATCH_IPHC;
  } else {
  //se destination address global dispatch 0x04
  RIME_IPHC_BUF->dispatch = SICSLOWPAN_DISPATCH_IPHC_GLOB;
  }
  /*
   * Version, traffic class, flow label
   * If flow label is 0, compress it. If traffic class is 0, compress it
   * We have to process both in the same time as the offset of traffic class
   * depends on the presence of version and flow label
   */
  if(((UIP_IP_BUF->tcflow & 0x0F) == 0) &&
     (UIP_IP_BUF->flow == 0) && ((UIP_IP_BUF->vtc & 0x0F) == 0)) {
      /* version, flow label and traffic class can be compressed */
      /* compress (elide) all */
      PRINTF("compress version traffic and flow \n");
      RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_VF_C;
   } else {
      /* compress nothing */
      PRINTF("no compress version traffic and flow \n");
      memcpy(hc00_ptr, &UIP_IP_BUF->vtc, 4);
      hc00_ptr += 4;
   }
  /* Note that the payload length is always compressed */
  /* Next header. We compress it if UDP */
#if UIP_CONF_UDP
  if(UIP_IP_BUF->proto == UIP_PROTO_UDP) {
    RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_NH_C;
  } else {
#endif /*UIP_CONF_UDP*/
     *hc00_ptr = UIP_IP_BUF->proto;
      hc00_ptr += 1;
#if UIP_CONF_UDP
  }
#endif /*UIP_CONF_UDP*/
  // Hop limit always carry hop limit, like tinyOS
  *hc00_ptr = UIP_IP_BUF->ttl;
   hc00_ptr += 1;
  /* source address - cannot be multicast */
```

```
 if((context = addr_context_lookup_by_prefix(&UIP_IP_BUF->srcipaddr))
    != NULL) {
  /* elide the prefix */
  if(uip_is_addr_mac_addr_based(&UIP_IP_BUF->srcipaddr, &uip_lladdr)){
    /* elide the IID */
    RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_SAM_0;
  } else {
    if(sicslowpan_is_iid_16_bit_compressable(&UIP_IP_BUF->srcipaddr)){
      /* compress IID to 16 bits */
      RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_SAM_16;
      memcpy(hc00_ptr, &UIP_IP_BUF->srcipaddr.u16[7], 2);
      hc00_ptr += 2;
    } else {
      /* do not compress IID */
      RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_SAM_64;
      memcpy(hc00_ptr, &UIP_IP_BUF->srcipaddr.u16[4], 8);
      hc00_ptr += 8;
    }
  }
 } else {
  /* send the full address */
  RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_SAM_I;
  memcpy(hc00_ptr, &UIP_IP_BUF->srcipaddr.u16[0], 16);
  hc00_ptr += 16;
 }

 /* dest address*/
 if(uip_is_addr_mcast(&UIP_IP_BUF->destipaddr)) {
  /* Address is multicast, try to compress */
  if(sicslowpan_is_mcast_addr_compressable(&UIP_IP_BUF->destipaddr)) {
    RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_16;
    /* 3 first bits = 101 */
    *hc00_ptr = SICSLOWPAN_IPHC_MCAST_RANGE;
    /* bits 3-6 = scope = bits 8-11 in 128 bits address */
    *hc00_ptr |= (UIP_IP_BUF->destipaddr.u8[1] & 0x0F) << 1;
    /*
     * bits 7 - 15 = 9-bit group
     * We just copy the last byte because it works
     * with currently supported groups
     */
    *(hc00_ptr + 1) = UIP_IP_BUF->destipaddr.u8[15];
    hc00_ptr += 2;
  } else {
    /* send the full address */
    RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_I;
    memcpy(hc00_ptr, &UIP_IP_BUF->destipaddr.u16[0], 16);
    hc00_ptr += 16;
  }
 } else {
  /* Address is unicast, try to compress */
  if((context = addr_context_lookup_by_prefix(&UIP_IP_BUF->destipaddr)) != NULL) {
    /* elide the prefix */
    if(uip_is_addr_mac_addr_based(&UIP_IP_BUF->destipaddr, (uip_lladdr_t *)rime_destaddr)) {
      /* elide the IID */
      RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_0;
    } else {
      if(sicslowpan_is_iid_16_bit_compressable(&UIP_IP_BUF->destipaddr)) {
        /* compress IID to 16 bits */
        RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_16;
        memcpy(hc00_ptr, &UIP_IP_BUF->destipaddr.u16[7], 2);
```

```
        hc00_ptr += 2;
       } else {
        /* do not compress IID */
        RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_64;
        memcpy(hc00_ptr, &UIP_IP_BUF->destipaddr.u16[4], 8);
        hc00_ptr += 8;
       }
      }
    } else {
     /* send the full address */
     RIME_IPHC_BUF->encoding[0] |= SICSLOWPAN_IPHC_DAM_I;
     memcpy(hc00_ptr, &UIP_IP_BUF->destipaddr.u16[0], 16);
     hc00_ptr += 16;
    }
  }
  uncomp_hdr_len = UIP_IPH_LEN;

#if UIP_CONF_UDP
  /* UDP header compression */
  if(UIP_IP_BUF->proto == UIP_PROTO_UDP) {
    if(HTONS(UIP_UDP_BUF->srcport)  >= SICSLOWPAN_UDP_PORT_MIN &&
       HTONS(UIP_UDP_BUF->srcport)  <  SICSLOWPAN_UDP_PORT_MAX &&
       HTONS(UIP_UDP_BUF->destport) >= SICSLOWPAN_UDP_PORT_MIN &&
       HTONS(UIP_UDP_BUF->destport) <  SICSLOWPAN_UDP_PORT_MAX) {
      /* we can compress. Copy compressed ports, full chcksum */
      *hc00_ptr = SICSLOWPAN_NHC_UDP_C;
      *(hc00_ptr + 1) =
        (u8_t)((HTONS(UIP_UDP_BUF->srcport) -
                SICSLOWPAN_UDP_PORT_MIN) << 4) +
        (u8_t)((HTONS(UIP_UDP_BUF->destport) -
                SICSLOWPAN_UDP_PORT_MIN));
      memcpy(hc00_ptr + 2, &UIP_UDP_BUF->udpchksum, 2);
      hc00_ptr += 4;
    } else {
      /* we cannot compress. Copy uncompressed ports, full chcksum */
      *hc00_ptr = SICSLOWPAN_NHC_UDP_I;
      memcpy(hc00_ptr + 1, &UIP_UDP_BUF->srcport, 4);
      memcpy(hc00_ptr + 5, &UIP_UDP_BUF->udpchksum, 2);
      hc00_ptr += 7;
    }
    uncomp_hdr_len += UIP_UDPH_LEN;
  }
#endif /*UIP_CONF_UDP*/
  rime_hdr_len = hc00_ptr - rime_ptr;
  return;
}
```

Di seguito si riporta il codice modificato per la funzione di decompressione in sicslowpan.c:

```
static void
uncompress_hdr_hc00(u16_t ip_len) {
  hc00_ptr = rime_ptr + rime_hdr_len + 2;

  /* Version and flow label */
  if((RIME_IPHC_BUF->encoding[0] & 0x80) == 0) {
    /* Version and flow label are carried inline AND CLASS !!! */
    memcpy(&SICSLOWPAN_IP_BUF->vtc, hc00_ptr, 4);
    hc00_ptr += 4;
```

```
  } else {
    /* Traffic class is compressed AND VERSION AND FLOW LABEL*/
    SICSLOWPAN_IP_BUF->vtc = 0x60;
    SICSLOWPAN_IP_BUF->tcflow = 0;
    SICSLOWPAN_IP_BUF->flow = 0;
  }
  /* Next Header */
  if((RIME_IPHC_BUF->encoding[0] & 0x40) == 0) {
    /* Next header is carried inline */
    SICSLOWPAN_IP_BUF->proto = *hc00_ptr;
    hc00_ptr += 1;
  }
  /* Hop limit */
  switch(RIME_IPHC_BUF->encoding[0] & 0x20) {
    case SICSLOWPAN_IPHC_TTL_255:
      SICSLOWPAN_IP_BUF->ttl = 255;
      break;
    case SICSLOWPAN_IPHC_TTL_I:
      SICSLOWPAN_IP_BUF->ttl = *hc00_ptr;
      hc00_ptr += 1;
      break;
  }
  /* Source address */
  // local-link context
  context =
    addr_context_lookup_by_number(0);
  //da 0 a 1 per test aaaa, only local-link context, verifica dispatch
  //in caso dispatch global 0x04
  if(RIME_IPHC_BUF->dispatch==SICSLOWPAN_DISPATCH_IPHC_GLOB){
  context = addr_context_lookup_by_number(1);
  }
  switch(RIME_IPHC_BUF->encoding[0] & 0x18) {
    case SICSLOWPAN_IPHC_SAM_0:
      if(context == NULL) {
        PRINTF("sicslowpan uncompress_hdr: error context not found\n");
        return;
      }
      /* copy prefix from context */
      memcpy(&SICSLOWPAN_IP_BUF->srcipaddr, context->prefix, 8);
      /* infer IID from L2 address */
      uip_netif_addr_autoconf_set(&SICSLOWPAN_IP_BUF->srcipaddr,
                                  (uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_SENDER));
      break;
    case SICSLOWPAN_IPHC_SAM_16:
      if((*hc00_ptr & 0x80) == 0) {
        /* unicast address */
        if(context == NULL) {
          PRINTF("sicslowpan uncompress_hdr: error context not found\n");
          return;
        }
        memcpy(&SICSLOWPAN_IP_BUF->srcipaddr, context->prefix, 8);
        /* copy 6 NULL bytes then 2 last bytes of IID */
        memset(&SICSLOWPAN_IP_BUF->srcipaddr.u8[8], 0, 6);
        memcpy(&SICSLOWPAN_IP_BUF->srcipaddr.u8[14], hc00_ptr, 2);
        hc00_ptr += 2;
      } else {
        /* multicast address check the 9-bit group-id is known */
        if(sicslowpan_is_mcast_addr_decompressable(hc00_ptr)) {
          SICSLOWPAN_IP_BUF->srcipaddr.u8[0] = 0xFF;
          SICSLOWPAN_IP_BUF->srcipaddr.u8[1] = (*hc00_ptr >> 1) & 0x0F;
```

```
        memset(&SICSLOWPAN_IP_BUF->srcipaddr.u8[2], 0, 13);
        SICSLOWPAN_IP_BUF->srcipaddr.u8[15] = *(hc00_ptr + 1);
        hc00_ptr += 2;
       } else {
        PRINTF("sicslowpan uncompress_hdr: error unknown compressed mcast address\n");
        return;
       }
     }
     break;
   case SICSLOWPAN_IPHC_SAM_64:
     if(context == NULL) {
      PRINTF("sicslowpan uncompress_hdr: error context not found\n");
      return;
     }
     /* copy prefix from context */
     memcpy(&SICSLOWPAN_IP_BUF->srcipaddr, context->prefix, 8);
     /* copy IID from packet */
     memcpy(&SICSLOWPAN_IP_BUF->srcipaddr.u8[8], hc00_ptr, 8);
     hc00_ptr += 8;
     break;
   case SICSLOWPAN_IPHC_SAM_I:
     /* copy whole address from packet */
     memcpy(&SICSLOWPAN_IP_BUF->srcipaddr.u8[0], hc00_ptr, 16);
     hc00_ptr += 16;
     break;
 }

 /* Destination address */
 // local-link context
 context = addr_context_lookup_by_number(0);
 // da 0 a 1 per test aaaa, only local-link context, verifica dispatch
 //in caso dispatch global 0x04
 if(RIME_IPHC_BUF->dispatch==SICSLOWPAN_DISPATCH_IPHC_GLOB){
 context = addr_context_lookup_by_number(1);
 }
 switch(RIME_IPHC_BUF->encoding[0] & 0x06) {
   case SICSLOWPAN_IPHC_DAM_0:
     if(context == NULL) {
      PRINTF("sicslowpan uncompress_hdr: error context not found\n");
      return;
     }
     /* copy prefix from context */
     memcpy(&SICSLOWPAN_IP_BUF->destipaddr, context->prefix, 8);
     /* infer IID from L2 address */
     uip_netif_addr_autoconf_set(&SICSLOWPAN_IP_BUF->destipaddr,
                          (uip_lladdr_t *)packetbuf_addr(PACKETBUF_ADDR_RECEIVER));
     break;
   case SICSLOWPAN_IPHC_DAM_16:
     if((*hc00_ptr & 0x80) == 0) {
      /* unicast address */
      if(context == NULL) {
       PRINTF("sicslowpan uncompress_hdr: error context not found\n");
       return;
      }
      memcpy(&SICSLOWPAN_IP_BUF->destipaddr, context->prefix, 8);
      /* copy 6 NULL bytes then 2 last bytes of IID */
      memset(&SICSLOWPAN_IP_BUF->destipaddr.u8[8], 0, 6);
      memcpy(&SICSLOWPAN_IP_BUF->destipaddr.u8[14], hc00_ptr, 2);
      hc00_ptr += 2;
     } else {
```

```
      /* multicast address check the 9-bit group-id is known */
      if(sicslowpan_is_mcast_addr_decompressable(hc00_ptr)) {
        SICSLOWPAN_IP_BUF->destipaddr.u8[0] = 0xFF;
        SICSLOWPAN_IP_BUF->destipaddr.u8[1] = (*hc00_ptr >> 1) & 0x0F;
        memset(&SICSLOWPAN_IP_BUF->destipaddr.u8[2], 0, 13);
        SICSLOWPAN_IP_BUF->destipaddr.u8[15] = *(hc00_ptr + 1);
        hc00_ptr += 2;
      } else {
        PRINTF("sicslowpan uncompress_hdr: error unknown compressed mcast address\n");
        return;
      }
    }
    break;
  case SICSLOWPAN_IPHC_DAM_64:
    if(context == NULL) {
      PRINTF("sicslowpan uncompress_hdr: error context not found\n");
      return;
    }
    memcpy(&SICSLOWPAN_IP_BUF->destipaddr, context->prefix, 8);
    memcpy(&SICSLOWPAN_IP_BUF->destipaddr.u8[8], hc00_ptr, 8);
    hc00_ptr += 8;
    break;
  case SICSLOWPAN_IPHC_DAM_I:
    /* copy whole address from packet */
    memcpy(&SICSLOWPAN_IP_BUF->destipaddr.u8[0], hc00_ptr, 16);
    hc00_ptr += 16;
    break;
 }
 uncomp_hdr_len += UIP_IPH_LEN;

 /* Next header processing - continued */
 if((RIME_IPHC_BUF->encoding[0] & 0x40) != 0) {
   /* The next header is compressed, NHC is following */
   if((*hc00_ptr & 0x10) == SICSLOWPAN_NHC_UDP_ID) {
     SICSLOWPAN_IP_BUF->proto = UIP_PROTO_UDP;
     switch(*hc00_ptr) {
       case SICSLOWPAN_NHC_UDP_C:
         /* 1 byte for NHC, 1 byte for ports, 2 bytes chksum */
         SICSLOWPAN_UDP_BUF->srcport = HTONS(SICSLOWPAN_UDP_PORT_MIN +
                                             (*(hc00_ptr + 1) >> 4));
         SICSLOWPAN_UDP_BUF->destport = HTONS(SICSLOWPAN_UDP_PORT_MIN +
                                              ((*(hc00_ptr + 1)) & 0x0F));
         memcpy(&SICSLOWPAN_UDP_BUF->udpchksum, hc00_ptr + 2, 2);
         hc00_ptr += 4;
         break;
       case SICSLOWPAN_NHC_UDP_I:
         /* 1 byte for NHC, 4 byte for ports, 2 bytes chksum */
         memcpy(&SICSLOWPAN_UDP_BUF->srcport, hc00_ptr + 1, 2);
         memcpy(&SICSLOWPAN_UDP_BUF->destport, hc00_ptr + 3, 2);
         memcpy(&SICSLOWPAN_UDP_BUF->udpchksum, hc00_ptr + 5, 2);
         hc00_ptr += 7;
         break;
       default:
         PRINTF("sicslowpan uncompress_hdr: error unsupported UDP compression\n");
         return;
     }
     uncomp_hdr_len += UIP_UDPH_LEN;
   }
 }
```

```
  rime_hdr_len = hc00_ptr - rime_ptr;

  /* IP length field. */
  if(ip_len == 0) {
    /* This is not a fragmented packet */
    SICSLOWPAN_IP_BUF->len[0] = 0;
    SICSLOWPAN_IP_BUF->len[1] = packetbuf_datalen() - rime_hdr_len + uncomp_hdr_len -
UIP_IPH_LEN;
  } else {
    /* This is a 1st fragment */
    SICSLOWPAN_IP_BUF->len[0] = (ip_len - UIP_IPH_LEN) >> 8;
    SICSLOWPAN_IP_BUF->len[1] = (ip_len - UIP_IPH_LEN) & 0x00FF;
  }

  /* length field in UDP header */
  if(SICSLOWPAN_IP_BUF->proto == UIP_PROTO_UDP) {
    memcpy(&SICSLOWPAN_UDP_BUF->udplen, &SICSLOWPAN_IP_BUF->len[0], 2);
  }
  return;
}
```

Inoltre nel file sicslowpan.c è stato aggiunto il case per il dispatch globale nella funzione input():

```
#if SICSLOWPAN_CONF_COMPRESSION == SICSLOWPAN_CONF_COMPRESSION_HC00
    case SICSLOWPAN_DISPATCH_IPHC:
      PRINTF("sicslowpan input: IPHC\n");
      uncompress_hdr_hc00(frag_size);
      break;
     case SICSLOWPAN_DISPATCH_IPHC_GLOB:
      PRINTF("sicslowpan input: IPHC\n");
      uncompress_hdr_hc00(frag_size);
      break;
#endif /*SICSLOWPAN_CONF_COMPRESSION ==
SICSLOWPAN_CONF_COMPRESSION_HC00*/
```

Di seguito si riporta il codice modificato in sicslowpan.h:

```
#define SICSLOWPAN_DISPATCH_IPHC                0x03
#define SICSLOWPAN_DISPATCH_IPHC_GLOB           0x04
/*
* Values of fields within the IPHC encoding first byte
 * (C stands for compressed and I for inline)
 */
#define SICSLOWPAN_IPHC_TC_C                    0x80
#define SICSLOWPAN_IPHC_VF_C                    0x80
#define SICSLOWPAN_IPHC_NH_C                    0x40
#define SICSLOWPAN_IPHC_TTL_1                   0x20
#define SICSLOWPAN_IPHC_TTL_64                  0x20
#define SICSLOWPAN_IPHC_TTL_255                 0x20
#define SICSLOWPAN_IPHC_TTL_I                   0x00

/* Values of fields within the IPHC encoding second byte */
#define SICSLOWPAN_IPHC_SAM_I                   0x00
#define SICSLOWPAN_IPHC_SAM_64                  0x08
#define SICSLOWPAN_IPHC_SAM_16                  0x10
#define SICSLOWPAN_IPHC_SAM_0                   0x18
#define SICSLOWPAN_IPHC_DAM_I                   0x00
```

```
#define SICSLOWPAN_IPHC_DAM_64          0x02
#define SICSLOWPAN_IPHC_DAM_16          0x04
#define SICSLOWPAN_IPHC_DAM_0           0x06
```

## Appendice D: Comandi shell di Contiki 2.3

Contiki dispone anche di una shell, accessibile tramite USB, di seguito si riportano i comandi shell a disposizione:

| Comando | Descrizione |
|---|---|
| append <filename> | append to file |
| binprint | print binary data in decimal format |
| blink [num] | blink LEDs ([num] times) |
| broadcast | broadcast data to all neighbors |
| collect | collect data from the network |
| dec64 | decode base64 input |
| echo <text> | print <text> |
| format | format the flash-based Coffee file system |
| hd | print binary data in hexadecimal format |
| help | shows this help |
| kill <command> | stop a specific command |
| killall | stop all running commands |
| ls | list files |
| mac <onoroff> | turn MAC protocol on (1) or off (0) |
| netcmd <command> | run a command on all nodes in the network |
| nodeid | set node ID |
| nodes | get a list of nodes in the network |
| null | discard input |
| packetize | put data into one packet |
| ps | list all running processes |
| randwait <maxtime> <command> | wait for a random time before running a command |
| read <filename> [offset] [block size] | read from a file, with the offset and the block size as options |
| reboot | reboot the system |
| repeat <num> <time> <command> | run a command every <time> seconds |
| rfchannel <channel> | change CC2420 radio channel (11 - 26) |
| rm <filename> | remove the file named filename |
| routes | dump route list in binary format |
| send | send data to the collector node |
| sense | print out sensor data |
| senseconv | convert 'sense' data to human readable format |
| size | print the size of the input |
| sky-alldata | sensor data, power consumption, network stats |
| sniff | dump incoming packets |
| time [seconds] | output time in binary format, or set time in seconds since 1970 |
| timestamp | prepend a timestamp to data |
| txpower <power> | change CC2420 transmission power (0 - 31) |
| unicast <node addr> | unicast data to specific neighbor |
| write <filename> | write to file |

**Elenco delle figure**

**Bibliografia**

[1] Z. Shelby, C. Bormann “6LoWPAN The Wireless Embedded Internet”, Wiley

[2] RFC4294, “IPv6 Node requirements”

[3] RFC4944, “IPv6 over 6lowpan”

[4] J. Hui Arch Rock Corp. “Compression Format for IPv6 Datagrams in 6LoWPAN Networks draft-hui-6lowpan-hc-00”

[5] J. Hui Arch Rock Corp. “Compression Format for IPv6 Datagrams in 6LoWPAN Networks draft-hui-6lowpan-hc-01”

[6] A. Dunkels, B. Gronwall, T. Voigt, “Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors”

[7] M. Durvy, J. Abeillé, P. Wetterwald, C. O’Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, A. Dunkels, “Poster Abstract: Making Sensor Networks IPv6 ready”

[8] Thiemo Voigt, “Contiki COOJA Hands-on Crash Course: Session Notes” during 3rd Wide summer school on networked control system, Siena (IT)

[9] Crossbow web site: *http://www.xbow.com/*

[10] Mote*iv* web site: *http://moteiv.com/*

[11] Iana web site: *http://www.iana.org/assignment/6lowpan-parameters/6lowpan-parameters.xhtml*

[12] Contiki web site: *http://www.sics.se/contiki*