messaggio di controllo con FIN = 1 e client notificherà. Quando client chiede la chiusura il server notifica e quando avrà completato tutto l'invio dei messaggi invierà un messaggio di chiusura al client stesso e questo resterà in attesa in caso dovessero arrivare altri messaggi.

Per quanto riguarda capire come quando ci troviamo in situazione di congestione, dobbiamo capire con che velocità i segmenti verranno inviati in Rete poiché se mi rendo conto che ho congestione e continuo a inviare segmenti in modo veloce.
La quantità di byte da mandare N è calcolata considerando due valori (numero MAX di byte da trasmettere affinché non ci sia saturazione buffer, e numero MAX di byte da mandare per evitare congestione) da congestion window (cwnd) sarà data dal minimo di questi due valori)

Per capire se si ha congestione bisogna costruire algoritmi che ci aiutino a capirlo.
Abbiamo una situazione in cui nel nodo intermedio ho un buffer illimitato (il TCP non sfrutta velocità in base alla quantità di buffer libero dei nodi intermedi, ma se il buffer è pieno)
Se un nodo è collegato a un link con una capacità e la ricezione ne ha un'altra, la velocità con cui inviare i dati, non può superare il minimo delle due capacità.

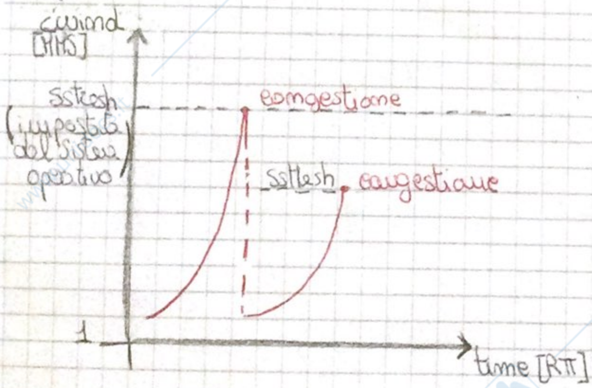
Quando nodo intermedio non ha spazio (limitato) per processare il pacchetto e inviarlo, questo sarà cancellato e non giungerà a destinazione. Bisogna ritrasmettere il messaggio.

Quando messaggio passa da livello applicativo a livello di trasporto si memorizza una copia nella coda comunque e cancello copia solo quando ho la conferma di ricezione.
Un altro problema nell'avere il buffer limitato è la velocità con cui i pacchetti escono è inferiore a quando entrano nel Router intermedio il pacchetto viene cancellato e quindi TCP deve rinviarlo e assicurare alla ritrasmissione un Time-Out, se alla fine di questo non ho ricevuto conferma vuol dire che ho avuto congestione e quindi ritrasmettere.
Come decido il TIME-OUT? Se lo setto troppo basso posso provocare congestione poiché invio troppi pacchetti in poco tempo, devo regolare quindi anche la velocità di invio pacchetti.
Per regolare la velocità dei segmenti in Rete utilizzo l'approccio END TO END:

Il TCP parte con il testare la capacità in Rete impostando la dimensione della finestra pari alla dimensione MAX di un segmento e ogni volta che riceve ACK di conferma, questo viene raddoppiato (crescita esponenziale o lineare) fino a quando non ho congestione. Si accorge della congestione quando si verificano due episodi:

- Scadere TIME-OUT (STOP AND WAIT)
- RICEVO 3 ACK DUPLICATI → quando il client invia i segmenti e riceve ACK, può capitare per esempio che invio ACK = 1 ACK = 2 ACK = 3 allora client invia in seguito 4, 5, 6 ma il server continuerà a mandare ACK = 3 (si aspetta di vedere il 3) e lo fa per 3 volte consecutive. Ovviamente il client non ritrasmetterà subito il segmento 3 poiché può capitare che questo arrivi più tardi. Ma se alla ricezione del 3 ACK vuol dire che c'è qualche problema (

Ci sono due algoritmi che controllano la velocità di trasmissione e differiscono solo per come reagiscono alla congestione (TCP RENO, TCP TAHOE)



Supponiamo che la dimensione della finestra sia pari alla congestion window e che la dimensione MAX del segmento da inviare in Rete [MMS] sia settata a 1.
Per cambiare la dimensione della finestra nel tempo il TCP utilizza AIMD, consiste nel trasmettere dati aumentando la velocità di trasmissione un po' alla volta (ADDITIVE) nel momento in cui il TCP si rende conto della congestione, la finestra si riduce drasticamente.

Abbiamo la congestion Window che parte da 1 e cresce nel tempo fino a arrivare a sstresh dove non ho ancora congestione ma devo stare attenta. Quando si presenta congestione, il TCP Tahoe per entrambi i casi (TIME OUT e TREE DUP ACK) prende il valore della finestra e la porta pari a 1 e anche la sstresh viene dimezzata e ricomincia il processo da capo.
Il TCP Reno invece affronta in un modo il Time Out e in un altro modo il 3dupack, associando loro un livello di importanza.
Nel Time-Out riduco a 1, mentre nel 3dupack non serve ridurre a 1 poiché la ricezione dei 3dupack può essere causata dal fatto che i pacchetti arrivano in ordine sparso quindi non si riduce drasticamente la finestra ma si dimezza.

- UDP protocollo poco affidabile poiché utilizzato per applicazioni poco affidabili. Riceve messaggio e lo invia a livello sottostante senza preoccuparsi se venga ricevuto, se si presenta di errore (non utilizza le funzioni del TCP/IP). L'obiettivo del protocollo UDP è implementare il MULTIPLEXING (Avere più flussi attivi in un unico Server, che provengono da client differenti) e DEMULTIPLEXING (quando arriva il

messaggio devo decidere a quale flusso appartiene). Funziona così:
UDP prende il messaggio, aggiunge il proprio header e lo consegna al livello di Rete, questo a sua volta aggiungerà un header, in cui troviamo anche un campo associato all'ID del pacchetto di trasporto.
Quando il nodo di destinazione si vede arrivare il pacchetto, legge ID del pacchetto di trasporto così da capire quale sia il pacchetto di destinazione. UDP non garantisce nessun [illegible] ordine ma è veloce

LIVELLO DI RETE → Abbiamo 4 aspetti fondamentali:
- DIMENSIONE DEL MESSAGGIO
- POSSIBILI PERCORSI PER ARRIVARE A DESTINAZIONE
- ROUTER INTERMEDIO IMPLEMENTA PRIMI 3 LIVELLI DELLO STACK

2 Protocolli importanti sono PROTOCOLLO ROUTING e IP

ROUTING: Responsabile della definizione del percorso all'interno della rete (CREA LA TABELLA ATTRAVERSO ALGORITMO DI ROUTING, ci sono regole per definire il percorso)

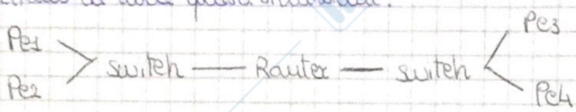
IP: Responsabile instradamento dei messaggi all'interno della Rete, interroga la tabella ma non la crea

PROTOCOLLO IP → contiene l'info principale che è l'indirizzo IP composto da 32 bit (4 byte), il quale identifica in modo univoco la rete.
L'indirizzo IP è definito con due soluzioni:
- SCRIVO A MANO
- UTILIZZO IL DHCP → quando PC non ha un IP e vuole comunicare con il Router, in questo è installato il DHCP, il quale prevede lo scambio di messaggi tra host, il Server che fornisce IP valido. 4 messaggi di controllo (2 quando sono implementati più server DHCP e 2 dove acquisisco e confermo IP)

- FRAMMENTAZIONE DEL MESSAGGIO → ogni singola tecnologia supporta una dimensione Max di messaggi (MTU). Quando si riceve un pacchetto superiore alla MTU si prende il pacchetto, lo si frammenta e lo si invia e il nodo di ricezione riassembla tutto.

Per implementare l'operazione di frammentazione e riassemblaggio abbiamo due valori da inserire nell'header (FLAGS e FRAGMENTS OFFSET)
Il fragment flags sarà settato a 1 quando voglio comunicare che quello è un frammento ma non è l'ultimo.
(OFFSET e FLAG = 0 non ho frammentazione) (FRAGS = 1 OFFSET = 0 primo segmento) (FLAGS = 0 OFFSET = 1 ho un ultimo frammento)

Ipotizziamo di avere questa situazione:



L'indirizzo IP è composto da 32 bit che divido in due parti.
La prima descrive la Rete dove il dispositivo è collegato, l'altra parte descrive interfaccia del dispositivo

Quando Pc1 e Pc2 vogliono comunicare è essenziale appartengano alla stessa rete, per capire se appartengono alla stessa rete, bisogna settare i byte dell'host = 0, se IP sono uguali vuol dire che fanno parte della stessa Rete. Ovviamente Pc1 deve conoscere IP di Pc2 e viceversa. → hanno stessa maschera
Una parte che compone l'IP è anche la maschera identificata come /24 e identifica i bit dedicati alla rete e i bit dedicati all'host (24 bit dedicati alla Rete e 8 all'host) → e fa capire se sono dentro la stessa Rete
Per ottenere indirizzo host bisogna settare a 0 la Rete
Abbiamo anche IP broadcast il quale si utilizza quando un nodo vuole comunicare con tutti i nodi all'interno della Rete e si fa settando l'ultimo byte = 1

192.4.15.0 → Rete
0.0.0.15 → Host
192.4.15.1 → Broadcast

Abbiamo bisogno di avere classi di IP poiché la Classe A utilizzati nelle Big Network, Classe C Reti locali, la differenza è nel numero di dispositivi connessi.

Il router al centro separa le due reti locali, ha due interfacce (sinistra e destra) il protocollo NAT permette al router della nostra rete di trasformare IP privato in IP pubblico, nella tabella NAT,con il quale la rete si interfaccia con il resto del mondo. Il NAT lavora pure sulle porte, ne sceglie una random. Quando messaggio esce dalla rete il Nat modifica IP sorgente e porta sorgente e li trasforma in pubblici e arriverà a destinazione. Nel momento in cui la destinazione risponde alla richiesta, la risposta contiene IP sorgente e porta ciò che prima era destinazione, IP di destinazione e porta è ciò che era prima nella sorgente. Il router una volta che riceve il messaggio, in assenza di NAT non sa a qualè di questi PC deve essere inoltrato il messaggio, grazie alla tabella il NAT capisce che il messaggio è rivolto a una porta specifica è un messaggio che deve essere indirizzato al nodo interno alla rete e quindi fa la traduzione inversa, cambia IP di destinazione e Porta di Destinazione. L'operazione è fatta in modo che l'utente non se ne accorga ma le info del NAT possono essere modificate manualmente anche da noi.

Per capire come il dispositivo richiede indirizzo MAC ci sono due casi, uno quando sono all'interno della stessa Rete e uno quando sono in due Reti diverse.

Sappiamo che quando due nodi vogliono comunicare è essenziale che conoscano il IP di sorgente e destinazione allora come prima cosa, per capire se sono all'interno della stessa rete, prendono un IP sorgente e destinazione e ci applicano la maschera se IP sono uguali appartengono alla stessa rete se no sono di due reti diverse.

In base a se siano dentro la Rete o no ci si comporta di conseguenza per trovare indirizzo MAC (questo con IPV4) utilizza ARP:

- Partendo dal fatto che conosco IP di destinazione, per trovare indirizzo MAC, invio un messaggio ARP, in quale contiene il mio IP, il mio MAC e IP di destinazione, in Broadcast. Il nodo che contiene quell'IP di destinazione si farà avanti e mi manderà MAC di destinazione con ARP reply (Non Mando che "conosciuto" MAC per arrivare a destinazione da mandare all'IP di destinazione, sarà l'interfaccia del link che riceverà il messaggio e lo inoltrerà)

IP SORGENTE = Sorgente

IP DESTINAZIONE = Destinazione

MAC SORGENTE = Sorgente

MAC DESTINAZIONE = Mac Interfaccia di Uscita del Router che riceverà il Messaggio

Dopo MAC sorgente sarà del MAC di destinazione e MAC destinazione sarà quello del mio Router

---

L'802.15.4 è il primo protocollo che permette di instaurare i sensori sui dispositivi (basso costo, bassa Potenza di trasmissione, bassa data Rate) tutti questi posti per IoT.

La Topologia di Rete dell'802 è identificata da due:

- STELLA = ho un nodo centrale (PAN COORDINATOR) che coordina la rete e la configura, quindi se il PAN si accende, si accenderanno anche gli altri dispositivi connessi si autoconfigurano
- PEER TO PEER = qualunque coppia di nodi comunica con una comunicazione Punto - Punto.

È essenziale che il PAN COORDINATOR sia un nodo FFD (nodo che implica capacità computazionali più elevate) e più funzionalità

Gli altri nodi possono essere RFD (nodi più semplici con meno funzionalità)

Queste due topologie vengono implementate a livello locale, se volessimo implementare una cella grande bisognerebbe unire queste topologie, e formare così dei Cluster, ovvero qui cluster sono coordinati da un FFD e quindi grazie ai Cluster, dispositivi possono comunicare anche a distanza.

L'accesso al mezzo è regolato da nel beacon e quindi possono avere:

- NOT BEACON ENABLE = Regolato con approccio CSMA/CA
- BEACON ENABLE = accesso regolato dal PAN COORDINATOR → abbiamo un intervallo di tempo in superframe, il quale all'esterno ha un beacon ossia un messaggio di controllo, ed è diviso in 3 parti. Una dedicata a CAP (accesso gestito da CSMA/CA) e una dedicata a CFP (Nodi che hanno ricevuto le concessioni possono comunicare)

La regola di accesso al mezzo è gestita da TSCH (Time Slotted Channel hopping) → abbiamo la divisione del Tempo in Slot e ogni Slot è dedicato a una coppia di nodi che comunicano utilizzando frequenze sempre diverse quindi anche diverse canali si utilizzano tutti i canali, seguendo quello da utilizzare. Abbiamo sempre un salto tra canali, se tra due nodi che comunicano ho un muro, il blocca il passaggio delle frequenze, per evitare questo si sposta il canale e si prova a trasmettere). Quindi questi salti devono essere concordati sia dalla sorgente sia dalla destinazione.

Il fatto di conoscere il canale di comunicazione e soprattutto quando due nodi comunicano, rende questo processo più robusto e si evitano soprattutto le collisioni.

Lo Stack della 802.15.4 è stato creato da un gruppo di lavoro chiamato 6tish il quale appunto creato per implementare in IoT:

LIVELLO APPLICATIVO → COAP

LIVELLO DI TRASPORTO → UDP

LIVELLO RETE → IPV6, RPL

LIVELLO DI ADATTAMENTO → 6TOP, 6LOWPAN → questo livello serve perché fatto che lo 802 non è un livello per dispositivi IoT e quindi esistono dei gruppi di lavoro che stanno studiando questo spazio livello per risolvere questo problema.

Partendo dal protocollo COAP, sostituisce HTTP in quanto è molto più semplice. Utilizza una comunicazione asincrona quindi il client fa richiesta al Server ma la comunicazione non avviene attiva, bensì quindi un protocollo semplice a essere affidabile se a livello di trasporto implementa UDP? Utilizza UDP

messaggio layer: quali possono essere di 4 categorie.
- CONFIRMBLE = client manda richiesta e si aspetta la risposta con aggiunta di ACK di risposta alla richiesta
- NON CONFIRMBLE = non ho ack di risposta né alla richiesta né al processo
- ACKNOWLEDGMENT = Ack
- RESET =

Come fa quindi il livello di trasporto del Client a capire che quella risposta è per quella specifica richiesta? se UDP non implementa queste funzioni?
Con l'approccio Piggy Beggings → questo utilizza dei token ossia dei numeri random che vengono aggiunti alla richiesta del client così quando il Server risponde, nella richiesta aggiunge questo token identificativo del Segmento e il Client capisce che quella risposta è per quella specifica richiesta avvenuta anche in passato.
Grazie ai token riusciamo a controllare anche le risposte avvenute in duplicato. Il protocollo COAP utilizza l'approccio STOP AND WAIT (invio richiesta e aspetto risposta per un tempo Time Out, se scade il tempo e la risposta non arriva devo procedere con il Reinvio del messaggio)

A livello di Rete abbiamo IPV6 che ha delle funzionalità diverse rispetto a IPV4, una di queste è che IPV6 non implementa il protocollo NAT poiché IoT ha bisogno di molti dispositivi da connettere e soprattutto accessibili da qualsiasi parte del mondo, per questo è necessario avere tutti IP pubblici e capiamo che 2^{32} combinazioni non bastano. Per questo vengono implementate 2^{128} combinazioni e un indirizzo esadecimale composto da 8 byte divisi in gruppi di 4.
Ci sono varie combinazioni per semplificare per esempio i 4 zeri si identificano con uno zero e così via.
In più non invia in Broadcast ma in MultiCast e Unicast. Vuol dire che in multicast invio a un gruppo di utenti per esempio 7 e unicast dico "di questi 7 voglio inviarlo a 3"!

RPL: Protocollo per IPV6 che non implementa Distance Vector ma ragiona allo stesso modo. Manifesta un modo di trovare la strada, creo un albero e lo tratto come se fosse topologia di Rete e da questo albero creo una strada migliore (INVIA MESSAGGIO A ROUTER E NODI VICINI e crea questo albero detto router di frontiera (ON DO DENTRO E FUORI LA RETE) invia il messaggio DIO a Router più vicini, il quale viene inviato dai Router più vicini per creare la DODAG. Il Router di frontiera per capire chi fa parte del dag deve ricevere un messaggio DAO. Così da capire chi è più conveniente in termini di costo (quanto sono distanti) quello con minor costo si collega alla DODAG.
PROATTIVO

Nei protocolli di adattamento abbiamo 6TOP e 6LOWPAN. Partendo da 6LOWPAN abbiamo detto che i pacchetti in IoT devono essere piccoli infatti si implementa un Segmento MAX di 127 byte, se dovesse arrivare un pacchetto più grande cosa si fa? 6LOWPAN identifica la funzione di frammentazione e riassemblaggio pacchetti e in più elimina gli header [illegible].
6TOP invece implementa le funzioni di controllo e configurazione dello scheduling.

DISPOSITIVI LORA → Tecnologia a lungo Raggio, semplice da usare ma trasmette pochi dati.
La Rete funziona così:
abbiamo dispositivi END NODE (LORA) che si collegano al network tramite gateway. Quando gli END NODE devono mandare dati, li sparano in aria e i gateway che si trovano in "conoscenza" raccolgono questi dati (END NODE NON SANNO QUANTI GATEWAY SONO PRESENTI), quindi sotto questo punto di vista è semplice poiché non bisogna instaurare una connessione tra LORA e Gateway. Dopo i gateway abbiamo il network server ossia entità che gestisce la rete e all'esterno [illegible] dei server application i quali ospitano dei Servizi come per esempio cancellare i pacchetti [illegible] che sono stati ricevuti (QUANDO END NODE SPARA DATI, VENGONO RICEVUTI DA PIÙ GATEWAY)!
Quindi ogni END NODE ATTRAVERSO GATEWAY comunica con APPLICATION SERVER, le regole sono utilizzate nel collegamento tra END NODE e gateway da lì in poi possiamo utilizzare qualsiasi tecnologia.
Possiamo distinguere 3 classi:
CLASSE A: identificata dal fatto che non è collegata a rete corrente ma ha una batteria e quindi ha un duty cycle basso vuol dire che i dispositivi si accendono solo quando si trasmette e si aspetta l'evento ricezione. La batteria quindi dura nel tempo

CLASSE B: Attuatori (dispositivi che ricevono info di controllo a Application Server), sono dotati di batteria quando non possono essere collegati a Rete elettrica

CLASSE C: Dispositivi collegati a corrente elettrica e sono sempre disposti a ricevere dati

- Se un nodo manda un MEX DIO ed ha un costo maggiore può comunicare ma non può entrare nella DODAG poiché deve interfacciarsi con i nodi di costo minore (COSTI ALTI COMPROMETTONO LA RETE)

ACCODAMENTO
THROUPUT
DNS
IOT
BLEUTHOOT

2 B 7 1
R – A – B – C

[illegible] Re [illegible]