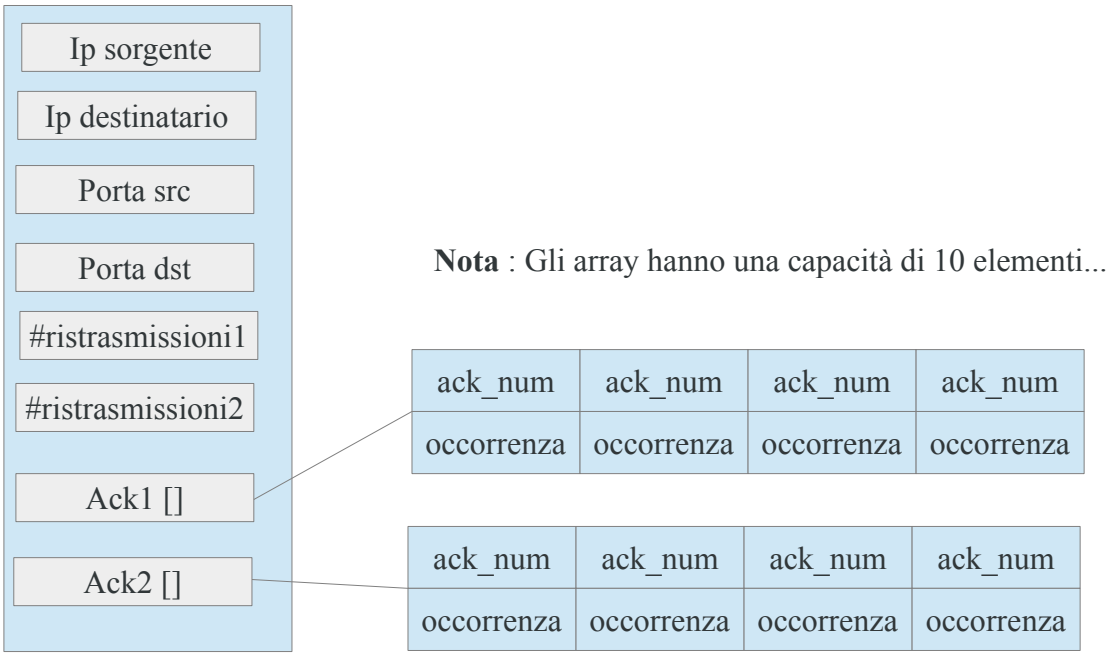
Descrizione Algoritmo Per L'Individuazione Delle Ritrasmissioni TCP

Mantengo per ogni connessione TCP, una struttura dati cosi fatta:



I primi 4 campi sono espliciti:si noti che con questa struttura consideriamo i pacchetti TCP che scorrono dalla sorgente verso la destinazione, ma anche il viceversa, infatti basta scambiare le sorgenti con le destinazioni. Ecco perché ho inserito due contatori #ritrasmissioni1 e ritrasmissioni2.

Ack1 e Ack2 sono due array: ogni elemento di questo array contiene un ack_number ossia la sequenza che richiede al pari e il numero di occorrenza di essa utile per capire se abbiamo una ritrasmissioni (>=4).

In breve: supponiamo che arrivi un pacchetto TCP p dall'host 1 . Questo può contenere o non contenere dati:

nel primo caso prendo la sua sequenza p.sequence e vado a vedere nell'array ACK2 se trovo questa sequenza e sopratutto se il numero di occorrenza è maggiore o uguale a 4, in tal caso incremento la variabile #ritrasmissione1. Se p.sequence non è presente non faccio niente.

Successivamente prendo in considerazione il campo p.ack_num. Andando a cercare questo valore nell'array ACK1: se è presente incremento il contatore (ACK1[i],occorrenza++),altrimenti aggiungo il nuovo valore e scarto il più vecchio ack_num che avevo nell'array.

Se invece p (a livello di TCP) non contiene dati, non può essere una ritrasmissione e quindi non considero p.sequence e passo subito a considerare il campo p.ack_num come descritto sopra..

Per un pacchetto pp proveniente dall'host 2 la procedura è analoga: si cerca pp.sequence in ACK1 e si aggiorna ACK2 con il valore di pp.ack_number.

Questo algoritmo è implementato nel metodo `update(TCPPacket p)` della calsse ConnssioneTCP

Questa struttura è definita dalla classe ConnessioneTCP e è mantenuta in un albero auto-bilanciante (classe AVL), in modo da ottenere una complessità logaritmica per l'operazione di inserzione.

Ogni radice di un AVL è mantenuta in una tabella hash, la cui chiave è la somma dell'indirizzo ip destinazione, sorgente, porte di destinazione e sorgente.

Ogni volta che arriva un pacchetto TCP:

```
private void insert(TCPPacket tcp) {

            ConnessioneTCP c = new ConnessioneTCP(tcp);
            Long key = new Long(c.getKey());
            AVL tmp;
            if ((tmp = (AVL) table.put(key, foo)) == null) {
                  AVL root = new AVL();
                  root.add(c, tcp);
                  table.put(key, root);
            } else {
                  tmp.add(c, tcp);
                  table.put(key, tmp);
            }

      }
```

Viene invocato il metodo insert: calcola la chiave per il pacchetto ricevuto, esegue il metodo put che restituisce il puntatore alla radice di un albero. Se questa è null, non ho collisioni, quindi posso creare un nuovo albero (root) provvedendo a inserirlo con il metodo put, rimuovendo foo. Altrimenti ho una collisione e in tmp ho il puntatore alla radice dell'albero, su cui invoco la add. Alla fine di questa funzione ho inserito o modificato un oggetto di tipo ConnessioneTCP nell'albero, adesso posso rimuovere l'oggetto foo e al suo posto rimettere l'albero.