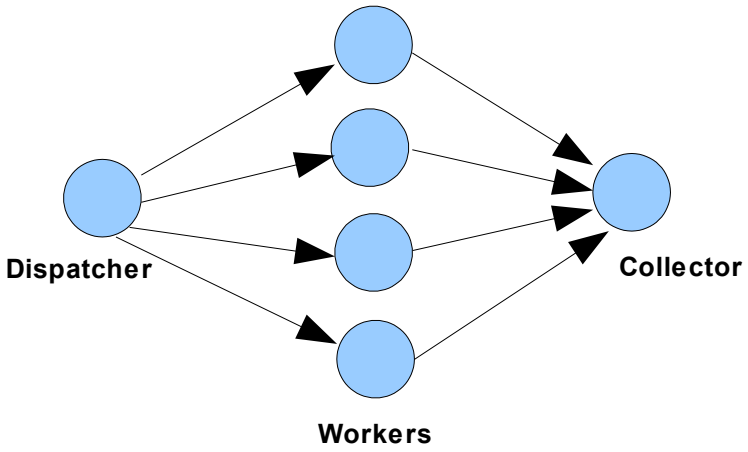
PKTFLOW versione 1.0 (file aggiornato)

L'applicazione e' realizzata secondo un modello architetturale parallelo di tipo farm:



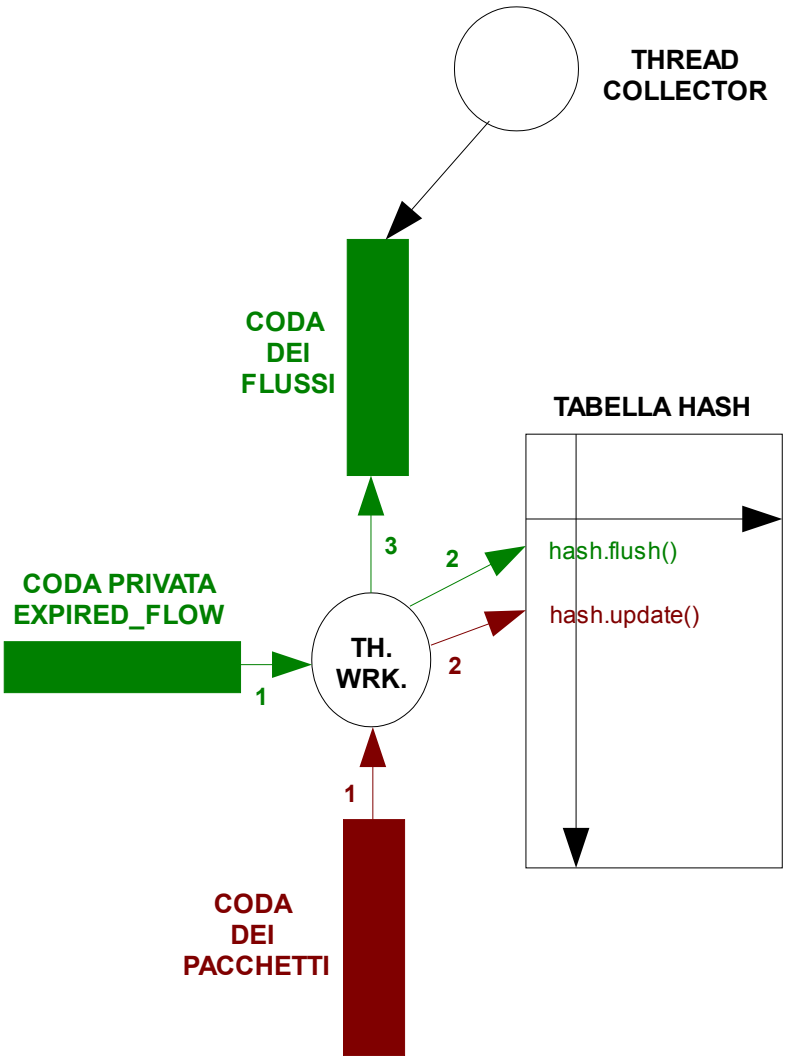
Il thread Dispatcher cattura pacchetti dai livelli sottostanti del sistema utilizzando le funzionalita' messe a disposizione dalla libreria pcap e li inoltra ai thread Worker.

I thread Worker prendono i pacchetti dalla coda a loro associata, calcolano il flusso relativo il pacchetto e lo inseriscono in una tabella hash comune a tutti i workers. Una volta che i flussi raggiungono il loro tempo di vita massimo, vengono rimossi dalla tabella hash e inoltrati ad un Collector attraverso una coda di flussi unica per ogni worker.

Il thread Collector raccoglie i flussi “spirati”da tutte le code a lui associate e stampa a video le informazioni.

Considerazioni riguardo i Worker:
Partendo dalle ottimizzazioni viste a lezione sappiamo che le inserzioni e gli aggiornamenti dei flussi non generano problemi di concorrenza. Per risolvere il problema della rimozione ho pensato di utilizzare una coda di expired_flow privata per ogni worker. Per ogni iterazione il worker fa l'hash.update() di TUTTI I PACCHETTI che trova nella coda dei pacchetti, una volta che la coda si e' svuotata controlla UN FLUSSO della coda degli expired_flow. Se il flusso e' “spirato” lo rimuove dalla tabella hash tramite la hash.flush(), lo invia al Collector e comincia una nuova iterazione (controlla quindi di nuovo tutti i pacchetti eccetera eccetera...)

Sotto e' presentato uno schema semplificato del funzionamento del worker.



1)Finche' ci sono pacchetti nella coda a loro associata fa l'hash.update() di TUTTI I PACCHETTI ricevuti, creando o aggiornando un flusso esistente

2)Controlla la coda dei flussi expired, SE UN FLUSSO e' expired perche' o ha raggiunto il tempo massimo di vita o il tempo massimo di inattivita' il flusso viene cancellato dalla tabella hash(hash.flush()), rimosso dalla coda dei flussi e inviato al Thread Collector attraverso una coda di flussi dedicata tra il singolo worker e Collector

3)Se non e' stata effettuata alcuna attivita' nell'iterazione corrente il thread va in attesa attiva

Considerazioni sulla coda expired_flow:

1. *Come vengono inseriti i flussi nella coda `expired_flow`?*
   I flussi vengono inseriti in coda un fase di creazione.
   Quando il worker(eseguendo `hash.update()` di un pacchetto) crea un nuovo flusso, inserisce il RIFERIMENTO al flusso appena creato in fondo alla coda `expired_flow` (FIFO)
2. *Qual'e' la relazione temporale dei flussi presenti in coda?*
   Partendo dal presupposto che solo il Dispatcher catturi i pacchetti da pcap, sappiamo che i timestamps dei pacchetti ricevuti dai Workers sono in ORDINE NON DECRESCENTE. Se un worker riceve 5 pacchetti e tutti quanti hanno chiavi diverse, alla fine di tutti gli `hash.update()` la coda avra' ordine:
   (K5,T5)->(K4,T4)->(K3,T3)->(K2,T2)->(K1,T1).
   Dove (Ki,Ti) indica un flusso avente chiave Ki e l'ultimo pacchetto ricevuto al tempo Ti
   Dove Ki e' una chiave distinta per ogni flusso.
   Dove Ti e' il timestamp dell'i-esimo del pacchetto ricevuto dal worker, tale che Ti<=Ti+1
3. *Cosa succede ai flussi presenti in coda in caso di aggiornamento di un flusso esistente?*
   Sapendo che i pacchetti ricevuti dal worker hanno un ordinamento temporale preciso, e' chiaro che in caso di aggiornamento di un flusso, per mantenere l'ordinamento temporale

in `expired_flow`, il flusso deve essere re-inserito in fondo alla coda.
Esempio: arriva un pacchetto avente chiave K2 al tempo T6; la coda `expired_flow` deve essere cambiata in maniera tale che da:
(K5,T5)->(K4,T4)->(K3,T3)->**(K2,T6)**->(K1,T1).
Diventi:
(K2,T6)->(K5,T5)->(K4,T4)->(K3,T3)->(K1,T1).
Problema: Come rimuove un flusso e re-accodarlo a tempo O(1)? Se devo scorrere la coda per trovare il flusso, la complessita' dell'`hash.update()` schizza a O(n), perche' per ogni aggiornamento si deve pure aggiornare la coda degli expires.
La soluzione che ho trovato (per risparmiare tempo e spazio) e' quella di implementare flussi "puntati".
Ogni oggetto Flow ha tre puntatori:

1. puntatore al prossimo flusso nella tabella hash (in caso di collisioni hash)
2. puntare al flusso successivo presente nella coda `expired_flow`
3. puntatore al flusso precedente presente nella coda `expired_flow`

Ossia la coda e' semplicemente composta da quei flussi gia' presenti in tabella hash che puntano al flusso precedente e al successivo della coda stessa.
In questa maniera posso "rimuovere" un flusso da una posizione qualsiasi della coda `expired_flow` a tempo O(1), in quanto non faccio altro che far puntare il suo precedente al suo successivo e il suo successivo al suo precedente.

4. *Come vengono rimossi i flussi dalla coda `expired_flow`?*
   Una volta che il Worker ha eseguito gli `hash.update()` di tutti i pacchetti disponibili, controlla il flusso piu' vecchio presente in coda `expired_flow`(FIFO).Se il flusso e' spirato perche' o ha raggiunto il suo tempo massimo di vita o ha raggiunto il suo tempo massimo di inattivita', lo rimuove sia dalla tabella hash (`hash.flush()`) sia dalla coda `expired_flow` che diventa:
   (K2,T6)->(K5,T5)->(K4,T4)->(K3,T3) ...(K1,T1) <-flusso piu' vecchio rimosso
   Infine invia il flusso al Collector.

Lo spazio occupato in memoria dalla coda `expired_flow` e' trascurabile in quanto coincide sostanzialmente con quello dei flussi gia' presenti in tabella hash. I tempi di accesso dovrebbero essere O(1) in quanto le tre operazioni che mette a disposizione sono:

1. **`addLast(Flow)`**:O(1) in fase di creazione di un nuovo flusso.
2. **`removeFirst()`**:O(1) se il flusso piu' vecchio e' spirato viene rimossa la testa.
3. **`remove(Flow)`**:O(1) se il flusso deve essere nuovamente accodato in caso di aggiornamento non faccio altro che aggiornare i suoi puntatori al precedente e al successivo in coda e lo re-accodo con **`addLast(Flow)`**.

A mio avviso l'impatto della coda `expired_flow` sulle prestazioni dei worker e' minimo dato che ogni iterazione il Worker fa l'update di TUTTI i pacchetti disponibili e al piu' la rimozione di UN SOLO flusso. Inoltre in fase di update il costo di inserzione o aggiornamento coda e' O(1) e quindi trascurabile in fase di esecuzione update stesso.
Infine il lavoro di visualizzazione flusso viene svolto dal thread Collector che raccoglie tutti i flussi spirati da tutti i Worker e li stampa a video; questo evita ai Worker di fare lavoro inutile.
L'utilizzo del Collector non dovrebbe costituire un collo di bottiglia in quanto tutti i worker comunicano con il Collector utilizzando la loro coda dedicata.
Esiste certamente una possibilita' molto concreta di "starving". Infatti se la sonda cattura moltissimi pacchetti a causa di un traffico molto elevato all'interno della rete, e' molto probabile che il worker cicli in continuazione eseguendo gli hash updates senza mai entrare nella parte di codice relativa alla rimozione dei flussi.
Ma nel progetto e' stata posta la massima importanza riguardo la creazione e l'aggiornamento di

flussi(se non si e' veloci si perdono pacchetti), mentre al contrario la tematica della rimozione ha una rilevanza minore.
In ogni caso il problema si puo' risolvere con una politica di “quasi fairness”. Ossia se sto facendo troppi updates, mi fermo un attimo casomai ho qualche flusso da rimuovere.

Considerazione sulle code:
Un ultima considerazione riguarda le code utilizzate per far comunicare i vari threads tra di loro. Per una questione di efficienza sono state scritte ad “hoc”.

1. *`PacketQueue`*: Realizza una coda lock-free di dimensione finita per lo scambio di pacchetti tra il thread Dispatcher e un thread Worker. Per risolvere i problemi di sincronizzazione ho ulitizzato gli strumenti messi a disposizione da `java.util.concurrent`, in particolare la tecnologia “CAS”.
2. *`FlowQueue`*: Realizza una coda dinamica lock-free di flussi per la comunicazione tra il thread Collector e un thread Worker. L'implementazione della coda non ha manifestato alcuna necessita' di utilizzare tecnologia particolare per realizzare sincronizzazione lock-free tra Collector e Worker.

Per sfruttare a pieno i vantaggi computazionali offerti dall'utilizzo di queste code, tutti i threads dell'applicazione Packet Flow adottano una politica di attesa attiva, in quanto si presume che l'applicazione sia per la maggior parte del tempo sotto stress, dovuta alla cattura dei pacchetti.