Università di Padova - DEI
Corso “Design of High Speed Network Systems”, Anno Accademico 2002-2003

Term paper

Località, network processor e routing table lookup problem

Studente: Carlo Fantozzi, XVI ciclo

1 Contenuti del term paper

Questo term paper ha l’obiettivo di illustrare come la località presente nel traffico di rete possa essere impiegata per velocizzare la consultazione di una tabella di routing (*routing table lookup problem*). Nella Sezione 2 verrà introdotto il concetto di località, sia in termini generali che con specifico riferimento alle reti di comunicazione, e si parlerà di alcuni studi che hanno misurato la “quantità di località” presente nel traffico di rete. Nella Sezione 3 si discuterà come la crescita della velocità di trasmissione esiga la progettazione di dispositivi di rete sempre più veloci: l’utilizzo di cache (che sfruttano la località) e la costruzione di network processor vanno incontro a questa esigenza. Si illustrerà poi il lavoro di alcuni ricercatori dell’Università di Stony Brook, che hanno studiato come costruire cache per network processor allo specifico scopo di velocizzare la consultazione di tabelle di routing.

2 Il concetto di località nelle reti

Lo sfruttamento della località (*locality*) nell’accesso ai dati ha consentito di incrementare le prestazioni di molti sistemi informatici tra loro diversi. La località è stata studiata e proficuamente utilizzata prima relativamente all’accesso a memoria secondaria, poi nell’accesso ai file (attività comune ad esempio nella consultazione di una base di dati), poi nell’accesso alla memoria primaria (introduzione di più livelli di memorie cache) e infine nella gestione delle reti informatiche di comunicazione. Prima di esaminare più in dettaglio la questione relativa alle reti è però opportuno introdurre un po’ di terminologia.

Sono stati identificati due tipi fondamentali di località: località spaziale e località temporale. Con località temporale (*temporal locality*) si indica l’accesso ad un numero limitato di dati in un intervallo di tempo definito, mentre con località spaziale (*spatial locality*) si intende l’accesso a dati consecutivi in istanti di tempo consecutivi. In letteratura esistono riferimenti a molti concetti affini ai due appena presentati, con termini come “persistence” o, con specifico riferimento alle reti [9], “concentration” e “reference density”.

Definizione 2.1 *Si definisce concentrazione (*concentration*) di una sequenza di $w > 1$ pacchetti consecutivi la quantità $C_w := (w - W_w)/(w - 1)$, dove W_w è il numero di indirizzi distinti che compaiono nella sequenza.*

La concentrazione è una quantità che varia tra 0 (nessuna concentrazione) e 1 (concentrazione massima: un solo indirizzo compare nella sequenza), e può essere considerata una misura della località temporale.

Definizione 2.2 *Si definisce densità di accesso (*reference density*) di una sequenza di w pacchetti la quantità $R_{n,w} := A_n/w$, dove A_n è il numero di indirizzi distinti usati nella sequenza dagli n host più attivi sulla rete.*

In altre parole, la densità di accesso indica quale frazione degli indirizzi è utilizzata dagli n host più attivi sulla rete. Questa definizione ha minore attinenza con il concetto generale di località, e misura una caratteristica specifica delle reti.

Come comincia ad apparire ormai chiaro, per le reti di calcolatori la località di interesse in molte applicazioni è quella presente nel traffico di rete e relativa all'utilizzo degli indirizzi per i livelli 1 (fisico) e 3 (network) della pila ISO OSI: in altre parole, per "località" si intende il ricomparire più volte del medesimo indirizzo nel traffico che attraversa un punto fissato della rete di comunicazione. Frequentemente, tale punto coincide con un nodo della rete fisicamente occupato da un dispositivo quale un bridge, uno switch o un router che deve elaborare il traffico che lo attraversa; l'esistenza di località nel traffico permette di archiviare in memoria cache le strutture dati relative agli indirizzi che si presentano più frequentemente, aumentando così le prestazioni del dispositivo. Si noti che esiste una sostanziale differenza tra lo studio della località nelle reti di interconnessione ad altissime prestazioni per calcolatori paralleli e lo studio della località in reti informatiche pubbliche.

- Nel primo caso il traffico è frutto di un solo o di un numero limitato di utenti e può essere modificato dagli utenti stessi (tramite modifiche al programma che viene eseguito). Lo studio concerne la misurazione delle prestazioni corrispondenti alla topologia di rete di interesse e al successivo sviluppo di programmi utente che massimizzino la località con riferimento ai parametri misurati.

- Nel secondo caso il traffico è frutto di moltissimi utenti e non può essere modificato. La località è una caratteristica intrinseca del traffico e viene sfruttata per velocizzare (tramite caching) l'analisi e lo smistamento del traffico stesso.

In questo term paper ci si occuperà del secondo tipo di località.

L'importanza della località in reti pubbliche è apparsa chiara fin dalla metà degli anni Ottanta; numerosi studi ne hanno misurato l'entità [9, 10, 11, 12] secondo metriche diverse e suggerito possibili applicazioni [1, 5, 6, 9, 13]. Tali studi hanno utilizzato o registrazioni di traffico reale, o traffico reale rielaborato per eliminare fenomeni di regolarità statistica palesemente estranei al traffico stesso e dovuti a limiti nella presa dati (tempo di misura limitato, accesso al traffico di una porzione limitata di rete, o altro). Il risultato di questi studi, indipendentemente dalla metrica adottata, è che il traffico di rete esibisce una quantità significativa di località temporale dovuta all'esistenza di sessioni (interattive o di trasferimento dati) all'interno del flusso di pacchetti; non ha invece molto senso parlare di località spaziale nelle reti. Nonostante la presenza di località temporale, è più raro di quanto ci si aspetterebbe statisticamente che pacchetti consecutivi presentino lo stesso indirizzo: questo fenomeno viene spiegato con il fatto che il traffico è bidirezionale (se non altro a causa degli acknowledgement per ciascun pacchetto spedito), o più convincentemente con l'osservazione che il traffico è composto da più flussi dati multiplexati, e gli algoritmi di instradamento del traffico stesso sono di tipo "round robin". L'osservazione sulla bidirezionalità del traffico è comunque importante: se il traffico è bidirezionale ed esiste un pacchetto da A a B, allora con elevata probabilità sarà necessario gestire un pacchetto da B ad A e può quindi essere conveniente mantenere in cache sia l'indirizzo di A che quello di B. Alcuni ricercatori hanno recentemente osservato [15] che la "quantità di località" appare in diminuzione a causa del crescente numero di flussi dati che attraversano contemporaneamente una qualsiasi sezione di rete pubblica: servono quindi cache di dimensioni maggiori per compensare tale fenomeno. Oggi, la crescita della velocità di elaborazione non riesce a tenere il passo della crescita della velocità di trasmissione: lo sfruttamento della località temporale sta quindi diventando uno dei fattori imprescindibili per la costruzione di dispositivi che siano in grado di gestire un flusso di dati "alla velocità del cavo" (*at wire speed*), senza cioè costituire un collo di bottiglia nel sistema di comunicazione.

3 Routing table, cache e network processor

Un problema molto comune (e assai studiato in letteratura) in cui la crescita della velocità di trasmissione pone potenzialmente seri ostacoli è il cosiddetto "routing table lookup problem" (d'ora in avanti RTLP): dato un pacchetto di rete proveniente da una porta d'ingresso, si deve decidere a quale porta d'uscita inviarlo (*routing*) consultando le informazioni contenute in una *tabella di routing*. Il formato della tabella è variabile:

se il routing è effettuato al livello 3 della pila ISO OSI, allora la riga j-esima della tabella di routing contiene concettualmente un indirizzo IP di destinazione I_j, una maschera di rete (*network mask*) M_j e una porta d'uscita P_j; la maschera seleziona gli $|M_j|$ bit più significativi di I_j, e questi sono i bit effettivamente presi in considerazione durante il routing*. Dato un pacchetto IP in ingresso, sia D il relativo indirizzo di destinazione e sia k la riga della tabella di routing che contiene il più lungo prefisso di D: la porta d'uscita a cui inviare il pacchetto è allora P_k. La difficoltà di RTLP risiede nel numero estremamente elevato di pacchetti che devono essere esaminati ogni secondo. Ad esempio, da un canale da 1 Gbit/s può arrivare 1 milione di pacchetti da 1000 byte in un secondo, quindi non è possibile dedicare più di 1 microsecondo a ciascun pacchetto. Il problema assume dimensioni ancora maggiori se si considera che un qualsiasi router ha più porte d'ingresso e che il tempo di accesso per una memoria dinamica veloce è nell'ordine dei 50 nanosecondi: in questo scenario, basta una manciata di accessi in memoria per esaurire il tempo disponibile per il routing. Dato che RTLP è un problema chiave per la crescita sostenibile della rete globale Internet, esso è stato attaccato su molti fronti con lo scopo di velocizzare al massimo il processo di routing: si sono studiate strutture dati per memorizzare la tabella di routing in maniera compatta ed efficiente, si sono ideati algoritmi per velocizzare la consultazione della tabella stessa, si è proposto di utilizzare sistemi massicciamente paralleli per RTLP allo scopo di suddividere il carico di lavoro tra più processori; limitatamente a IPv4, sono state sviluppate tecniche "ad hoc" che velocizzano ulteriormente il routing facendo leva sulle peculiarità del protocollo IP. Tuttavia, tutto questo non è ancora sufficiente: negli ultimi anni l'overhead legato alla gestione del traffico di pacchetti (*packet processing*) è diventato così critico per le prestazioni che si è cominciato a sviluppare hardware dedicato a tale gestione. In questo contesto, i *network processor* sono un promettente tentativo di ottenere elevate prestazioni mantenendo almeno parte della flessibilità dei microprocessori tradizionali e con essa la capacità di adattarsi a mutamenti degli algoritmi e dei protocolli di rete.

Nel resto di questo term paper illustreremo alcuni lavori [2, 3, 8] presentati da ricercatori della State University of New York at Stony Brook: tali lavori si prefiggono di migliorare le prestazioni di RTLP per IPv4 sfruttando la località temporale presente nel traffico IP e memorizzando in una cache i dati (principalmente porzioni della tabella di routing) più frequentemente utilizzati durante il routing. Tale approccio non è naturalmente nuovo (si veda ad esempio [14, 16]): la novità risiede nella proposta di una architettura nello spirito di e adatta ai network processor, ovvero

- basata su una architettura di cache specializzata per RTLP;
- sufficientemente semplice da poter essere implementata con componenti già oggi presenti nei network processor.

Il resto di questa sezione è organizzato in tre sottosezioni corrispondenti ai contenuti dei tre lavori precedentemente citati. Nella Sezione 3.1 (vedi [2]) sarà descritta una struttura dati chiamata NART che consente di memorizzare la tabella di routing in forma compatta, facilitandone così il mantenimento in cache; si presenterà inoltre una strategia di utilizzo della cache chiamata HAC che non ha bisogno di alcun hardware speciale. Nella sezione 3.2 (vedi [3]) verranno invece presentate due strategie innovative chiamate HARC e IHARC che sfruttano il fatto che la cache non contiene dati generici ma indirizzi IP per aumentare la "copertura" dello spazio degli indirizzi e ridurre così il numero *cache miss*; tali strategie richiedono opportuno supporto da parte dell'hardware. Infine, nella sezione 3.3 (vedi [8]) vengono affrontate due potenziali difficoltà di HARC e IHARC, ovvero lo sbilanciamento del numero di indirizzi associati a pagine di cache diverse e l'overhead provocato da cambiamenti della tabella di routing; per ciascun problema vengono analizzate possibili soluzioni.

*Nel seguito si indicherà con il termine *netmask* l'insieme di bit identificati da un indirizzo IP I_j e dalla corrispondente maschera di rete M_j; tali bit verranno anche chiamati *bit significativi* di I_j.

3.1 NART e HAC

Un necessario prerequisito per risolvere efficientemente RTLP è memorizzare in maniera efficiente la tabella di routing, ovvero fare in modo che:

1. la tabella stessa occupi poco spazio in memoria;
2. sia possibile recuperare l'elemento della tabella associato ad un arbitrario indirizzo di destinazione D con un numero quanto più possibile limitato di accessi alla tabella stessa.

A questo proposito, si è da tempo compreso che cercare l'elemento della tabella associato a D equivale a cercare l'elemento k che, dopo l'applicazione della maschera M_k[†], fornisce il più lungo prefisso dell'indirizzo D. RTLP è stato così ricondotto al problema della ricerca di un prefisso di lunghezza massima: appare allora evidente che una struttura dati efficiente per la memorizzazione della tabella di routing è un *compacted trie* [4, 7] costruito sugli indirizzi che fanno parte della tabella, e in cui le foglie sono associate alle porte d'uscita; il massimo numero di accessi in memoria per RTLP è in questo caso pari all'altezza del trie. Purtroppo, per una miglior sfruttamento dei soli 2^{32} indirizzi IP disponibili è oggi in uso una tecnica chiamata *Classless Inter-Domain Routing* (CIDR) che consente di usare netsmak di qualsiasi lunghezza compresa tra 1 e 31 bit: anche l'altezza del trie è conseguentemente compresa in tale intervallo. Potrebbero dunque essere necessari decine di accessi alla memoria per per risolvere RTLP, e questo è inaccettabile. In [2], Chiueh e Pradhan propongono di risolvere questo problema costruendo un trie compresso su tre soli livelli, corrispondenti a netmask fino a 16 bit, da 17 a 24 bit e con 25 o più bit: a questa struttura dati viene dato il nome di *Network Address Routing Table* o NART. Più precisamente, la struttura NART prevede le tabelle seguenti.

- Una singola tabella Level-1 con 2^{16} elementi da 16 bit ciascuno, contenenti il numero di una porta d'uscita o un puntatore a una tabella Level-2. Ogni elemento è in corrispondenza biunivoca con una delle 2^{16} possibili configurazioni dei 16 bit più significativi di un indirizzo IP. Un netmask con 16 bit occupa un singolo elemento della tabella Level-1, che contiene il numero della corrispondente porta d'uscita; analogamente, un netmask con $L < 16$ bit occupa 2^{16-L} elementi della tabella Level-1, tutti contenenti il numero della corrispondente porta d'uscita. Se invece un netmask N contiene $L > 16$ bit allora l'elemento della tabella Level-1 corrispondente ai 16 bit più significativi di N contiene un opportuno offset ad una tabella Level-2.
- Zero, una o più tabelle Level-2 con 256 elementi da 8 bit ciascuno; gli elementi contengono il numero di una porta d'uscita o il codice speciale `0xFF`. Le tabelle Level-2 servono a gestire netmask di lunghezza compresa tra 17 e 24 bit: a tali netmask corrispondono tabelle Level-2 contenenti i numeri delle corrispondenti porte d'uscita. Se invece un netmask contiene più di 24 bit allora per essa viene comunque creata una tabella Level-2, ma gli elementi della tabella contengono il codice speciale `0xFF` che indica la necessità di consultare anche la tabella Level-3.
- Una singola tabella Level-3. Ciascun elemento di questa tabella è lungo 9 byte e contiene una riga completa della tabella di routing, comprendente quindi indirizzo IP, maschera di rete e numero di porta. La tabella Level-3 gestisce netmask di lunghezza maggiore di 24 bit; la ricerca di un elemento nella tabella richiede una scansione lineare.

Da quanto si è detto finora dovrebbe apparire chiaro che la ricerca del numero di porta corrispondente ad un indirizzo IP D richiede un accesso alla tabella Level-1 (usando i bit 16-31 di D), un eventuale accesso ad una opportuna tabella Level-2 (usando i bit 8-15 di D) e una eventuale scansione lineare della tabella Level-3 (usando anche i rimanenti bit di D). Non appena si incontra un elemento che contiene un numero di porta il procedimento di ricerca si arresta: di conseguenza, determinare la porta d'uscita per D richiede un

[†] Si ricordi che i bit non appartenenti alla maschera vengono ignorati.

solo accesso in memoria se il netmask che include D è lungo fino a 16 bit, e due soli accessi se tale netmask è lungo da 17 a 24 bit. I prefissi di lunghezza 16 e 24 bit costituiscono le classi A, B e C di indirizzi IP e comprendono la grande maggioranza delle sottoreti esistenti: l'eventualità di un accesso alla tabella Level-3 è statisticamente improbabile e, secondo gli autori, il corrispondente impatto sulle prestazioni di RTLP è trascurabile.

Una volta trovato il numero di porta corrispondente a D, il risultato della ricerca viene memorizzato in cache secondo la strategia *Host Address Cache* (HAC). Tale strategia è ispirata dall'osservazione che, almeno nei processori oggi più comuni, un indirizzo di memoria ha la stessa lunghezza di un indirizzo IP, ovvero 32 bit. Programmando opportunamente la MMU è dunque possibile salvare il numero di porta associato all'indirizzo D mediante una scrittura in memoria all'indirizzo virtuale D: in seguito alla scrittura il dato viene automaticamente portato in memoria cache, dove rimane disponibile per gli accessi successivi. Questa idea molto semplice, che può essere implementata con l'hardware disponibile in qualsiasi microprocessore moderno, si presta ad alcune ottimizzazioni che conducono alla strategia HAC.

- La cache dati è solitamente organizzata in *linee* la cui lunghezza è di qualche decina di byte: un accesso alla locazione di memoria x causa l'aggiornamento di una intera linea, con il caricamento delle locazioni adiacenti a x. Questa organizzazione è stata pensata per trarre vantaggio dalla località spaziale, che come si è detto non è però presente nel flusso di pacchetti di una rete (vedi Sezione 2). Di conseguenza, in questo caso è conveniente usare una singola linea di cache per memorizzare numeri di porta corrispondenti a indirizzi non adiacenti.

- Se l'hardware lo consente, si può riservare una porzione della cache esclusivamente alla gestione degli indirizzi IP mediante *locking* in cache di opportune pagine di memoria.

L'applicazione congiunta di queste osservazioni conduce ad un accesso misto hardware-software alla cache. Dato un indirizzo D, alcuni dei suoi bit vengono estratti per formare un opportuno indirizzo di memoria che, mediante il meccanismo di *cache lookup* del processore, individua una singola linea di cache; i rimanenti bit di D vengono impiegati come *tag* per determinare se, all'interno di tale linea, è presente l'indirizzo IP voluto con il corrispondente numero di porta. Ad esempio, si supponga di voler riservare per la strategia HAC $2^7 = 128$ linee di cache da $2^5 = 32$ byte ciascuna: in tal caso i bit 5-11 di ciascun indirizzo IP vengono utilizzati per l'accesso hardware alla cache, e i rimanenti bit sono poi impiegati per la scansione software della linea di cache così individuata. Per memorizzare i bit necessari alla ricerca nonché il numero di porta associato sono necessari 4 byte (sotto la ragionevole ipotesi che il numero di porte d'uscita sia inferiore a 128), quindi ogni linea di cache può contenere 8 coppie indirizzo-numero di porta.

A conclusione di questa sezione è bene riassumere il procedimento di soluzione per RTLP. Dato un indirizzo IP D, si consulta innanzitutto la cache secondo la strategia HAC: si costruisce dunque un opportuno indirizzo di memoria e si individua una linea di cache, dopodiché si ricerca D in tale linea. Se D è presente allora il procedimento si arresta, altrimenti si ricorre alla tabella NART: come si è visto ciò può richiedere uno, due o più accessi in memoria centrale. La cache viene poi aggiornata sulla base della ricerca nella tabella NART.

3.2 HARC e IHARC

In [3], Chiueh e Pradhan affinano la propria soluzione di RTLP proponendo due nuove strategie di gestione della cache. Entrambe le strategie hanno l'obiettivo di aumentare la "copertura efficace" (*effective coverage*) dello spazio degli indirizzi IP: in altre parole, esse si prefiggono di memorizzare informazioni su un maggior numero di indirizzi a parità di dimensione della cache, riducendo così l'eventualità di un cache miss. Come contropartita di una maggiore complessità, tali strategie richiedono supporto hardware specifico per essere implementate efficientemente. Tale supporto può comunque essere fornito combinando strutture molto

semplici, quali shifter e array di gate programmabili: è dunque ragionevole supporre che tali semplici strutture possano essere incluse in un network processor, ovvero un processore dedicato proprio all'elaborazione efficiente del traffico di rete.

La prima architettura di cache, che prende il nome di *Host Address Range Cache* (HARC), nasce dall'osservazione che ogni elemento di una tabella di routing associa la medesima porta d'uscita a un insieme di indirizzi IP consecutivi: per la precisione, se la netsmask è lunga L bit allora la corrispondente porta d'uscita è associata a 2^{32-L} indirizzi IP consecutivi. Questa osservazione suggerisce l'opportunità di inserire in memoria cache non il singolo indirizzo D oggetto dell'ultimo lookup, ma un intervallo di indirizzi contenente D e tale che gli indirizzi siano tutti associati alla medesima porta: se ciò può essere fatto "a basso costo", ovvero con un aggravio di tempo e memoria ridotti rispetto alla memorizzazione del singolo indirizzo D, allora l'efficienza della cache risulta notevolmente incrementata. È chiaramente desiderabile che gli intervalli di indirizzi siano quanto più ampi possibile, quindi conviene fondere anche più righe della tabella di routing in un unico insieme se tali righe riguardano indirizzi consecutivi e destinati alla medesima porta d'uscita. Un limite alla fusione è dato dalla presenza di netmask che specificano sottoinsiemi di altri netmask (*encompassed entries*). Ad esempio, la tabella di routing può specificare che tutti i pacchetti con indirizzo 147.162.*.* devono essere inviati alla porta P_i ad esclusione dei pacchetti 147.162.2.*, che devono invece essere inoltrati sulla porta $P_j \neq P_i$: l'insieme di 2^{16} indirizzi consecutivi specificato dal primo netmask viene così "spezzato" in due parti diseguali dal secondo netmask. Un limite di tipo diverso è dato dalla necessità di stabilire in maniera semplice e rapida i confini di ciascun intervallo di indirizzi: per ottenere ciò la strategia HARC esige che la dimensione di ciascun intervallo sia una potenza di 2. Riassumendo tutte queste osservazioni, l'individuazione degli intervalli di indirizzi può essere eseguita in due fasi.

1. Le righe della tabella di routing vengono fuse creando intervalli di indirizzi adiacenti la cui dimensione è massima compatibilmente con il vincolo che indirizzi nel medesimo intervallo devono essere associati alla stessa porta d'uscita.

2. Gli intervalli creati nella fase precedente vengono suddivisi in modo che la taglia di ciascun intervallo sia una potenza di 2.

La dimensione minima 2^m di un intervallo ottenuto dalla seconda fase viene chiamata *minima granularità* e dipende dal contenuto della tabella di routing: ad esempio, in [3] si afferma che la tabella di routing del progetto *Internet Performance Measurement and Analysis* (IPMA) fornisce $2^m = 32$. Poiché tutti gli intervalli hanno dimensione multipla di 2^m, gli m bit meno significativi di un indirizzo possono essere ignorati durante la consultazione della tabella di routing: in altre parole, ai fini del routing gli indirizzi IP sono ora descritti da $32 - m$ bit. Con la strategia HARC la copertura della cache risulta così incrementata di un fattore pari a 2^m. Per usare la memoria cache a piena velocità, i bit non significativi devono essere eliminati velocemente prima dell'accesso alla cache stessa: per questa ragione gli autori suggeriscono l'adozione di uno shifter che esegua in hardware uno shift logico a destra di m posizioni.

La seconda architettura di cache proposta in [3] prende il nome di *Intelligent Host Address Range Cache* (IHARC) e può essere considerata una naturale evoluzione di HARC. Con HARC viene identificato un intero m garantendo che gli m bit meno significativi degli indirizzi IP possano essere ignorati; durante il successivo accesso alla cache (che supponiamo contenere 2^k set) si usa come indice di linea un insieme di k bit consecutivi tra i $32-m$ più significativi. IHARC costituisce un miglioramento di HARC per due ragioni.

- IHARC sceglie i k bit da usare per l'accesso alla cache *prima* di determinare m, e li sceglie proprio con l'obiettivo di massimizzare m.

- IHARC è in grado di scegliere e utilizzare k bit *non consecutivi* per l'accesso alla cache.

È chiaro che l'utilizzo di bit non consecutivi per accedere alla cache richiede un supporto hardware specializzato: più precisamente, oltre allo shifter già visto per HARC è ora necessario un dispositivo in grado di

riorganizzare i bit di un indirizzo in modo da isolare i k bit che verranno usati come indice. Si noti che tale dispositivo deve essere programmabile, in quanto l'insieme dei k bit scelti varia al variare della tabella di routing.

Descriviamo ora IHARC in maggiore dettaglio. Come si è detto, il primo passo da compiere è quello di scegliere un insieme S di k bit per l'accesso alla cache massimizzando la dimensione degli intervalli di indirizzi. L'algoritmo di selezione proposto in [3] è di tipo greedy ed è descritto dallo pseudo-codice che segue.

$S \leftarrow \emptyset$
for $i \leftarrow 1$ **to** k **do**
    minscore $\leftarrow \infty$
    candidate $\leftarrow 0$
    for each $j \in \{0, 1, \ldots, 31\}$ not in S
        if $\mathrm{Score}(S \bigcup \{j\}) < \mathrm{minscore}$
            $\mathrm{minscore} \leftarrow \mathrm{Score}(S \bigcup \{j\})$
            candidate $\leftarrow j$
    $S \leftarrow S \bigcup \{\mathrm{candidate}\}$

L'algoritmo aggiunge all'insieme S un bit alla volta, selezionando ad ogni passo il bit che in quel momento risulta più "conveniente" sulla base della funzione di costo

$$\mathrm{Score}(S) = \sum_{i=1}^{2^{|S|}} M_i(S) + w \sum_{i=1}^{2^{|S|}} |\overline{M}(S) - M_i(S)| \tag{1}$$

determinata empiricamente. Per comprendere il significato di $\mathrm{Score}(S)$ si ricordi che i bit in S partizionano lo spazio degli indirizzi IP in $|S|$ insiemi disgiunti, ciascuno dei quali è associato ad una distinta configurazione dei bit; per ciascuno di tali insiemi è necessario determinare (come descritto per la strategia HARC) intervalli massimali di indirizzi associati alla medesima porta. Ebbene, $M_i(S)$ è il numero di intervalli nella partizione di indirizzi associata all'i-esima configurazione dei bit in S, e $\overline{M}(S)$ è il numero medio di intervalli calcolato considerando tutte le configurazioni:

$$\overline{M}(S) := \frac{1}{2^{|S|}} \sum_{i=1}^{|S|} M_i(S).$$

Il significato intuitivo della funzione di costo è dunque il seguente: il primo termine indica all'algoritmo di selezione dei bit di minimizzare il numero complessivo di intervalli di indirizzi, mentre il secondo termine gli indica di ripartire tali intervalli in maniera equilibrata tra le varie partizioni indotte da S. La quantità w è un peso scelto euristicamente per bilanciare il contributo dei due termini.

Una volta determinato l'insieme S di k bit, per ciascuna configurazione i dei rimanenti $32 - k$ bit si costruiscono gli intervalli massimali di indirizzi e si programma di conseguenza l'hardware necessario al funzionamento della cache. Si noti che la minima granularità m è ora funzione di i: come conseguenza, dati due cache set il numero di intervalli associati a ciascuno di essi è diverso. Esiste pertanto uno "sbilanciamento" nell'uso della cache, in quanto i cache set associati a un più alto numero di intervalli sono maggiormente soggetti a contention. Una ulteriore conseguenza dello sbilanciamento è che la strategia IHARC richiede alcuni adattamenti di natura tecnica della tabella NART.

Per valutare la bontà di HAC, HARC e IHARC gli autori forniscono in [3] i risultati di uno studio comparativo da essi compiuto e basato su una "traccia di pacchetti" (*packet trace*) appositamente raccolta. Più precisamente, la traccia è stata ottenuta registrando il traffico sulla linea T3 che collega il Brookhaven National Laboratory (stato di New York) alla rete pubblica Internet: per ciascuno dei circa 184 milioni di

pacchetti osservati dal 3 al 6 marzo 1998 si sono registrati l'indirizzo IP della sorgente, l'indirizzo della destinazione e un *timestamp*. Poiché il traffico osservato riguarda una porzione periferica della rete e può quindi potenzialmente esibire un grado di località superiore alla media, la traccia per gli esperimenti è stata ottenuta multiplexando i dati relativi a 100 intervalli temporali disgiunti della durata di 30 minuti ciascuno. Per la traccia così costruita è stato simulato il routing usando le tre strategie HAC, HARC e IHARC e variando alcuni parametri della cache quali la dimensione, il grado di associatività e la lunghezza di linea. Per la determinazione dei bit indice nella strategia IHARC si è posto w=1. I risultati delle simulazioni mostrano che, come prevedibile, il *miss ratio* diminuisce in maniera monotona all'aumentare della dimensione della cache e del grado di associatività. Il miss ratio diminuisce inoltre notevolmente al diminuire della lunghezza di linea, e raggiunge il minimo per linee di lunghezza unitaria: ciò dimostra che il traffico IP non presenta un livello significativo di località spaziale, e quindi non c'è alcun vantaggio nel caricare in cache indirizzi IP consecutivi a quello attualmente oggetto del routing. Per quanto riguarda il confronto tra le tre strategie, per una cache di 8 KB associativa a 2 vie HAC fornisce un miss ratio del $4,59\%$, HARC del $2,20\%$ e IHARC dello $0,48\%$.

3.3 Bilanciamento del carico e aggiornamento della tabella di routing

In [8], Gopalan e Chiueh riesaminano la strategia IHARC con l'obiettivo di migliorarne ulteriormente le prestazioni. L'esame rivela due possibili punti deboli della strategia.

1. Come si è visto nella sezione 3.2, il processo di selezione dei k bit indice (insieme S) può condurre a uno sbilanciamento nel numero di intervalli di indirizzi associati ai vari set della cache. I set associati a molti intervalli diventano potenziali "punti caldi" (*hot spots*) durante la consultazione della tabella di routing: essi possono infatti essere la causa di molti cache miss, riducendo così le prestazioni.

2. Ogni volta che la tabella di routing viene aggiornata, il contenuto di uno o più set della cache cessa di essere valido: per essere sicuri di non commettere errori nel routing è necessario invalidare l'intera cache. Finché non vengono caricate informazioni aggiornate si ha pertanto un fenomeno transitorio in cui la cache risulta inefficace e le prestazioni vengono sensibilmente penalizzate. Un aggiornamento della tabella di routing richiede inoltre, in base alla strategia IHARC, il ricalcolo dell'insieme di bit S, della minima granularità e della tabella NART, con conseguente non trascurabile overhead.

Relativamente al primo punto debole, in [8] vengono proposte due possibili soluzioni basate su due approcci completamente diversi, ciascuno dei quali presenta sia vantaggi che svantaggi. Il primo approccio mira a ridurre lo sbilanciamento scegliendo in maniera più accorta la funzione di pesatura $\mathrm{Score}(S)$: gli autori osservano infatti che il secondo addendo della funzione (1) non è una misura efficace dello sbilanciamento, come si può comprendere dal seguente esempio. Si supponga che, scegliendo S come insieme di bit indice, gli scostamenti di $M_i(S)$ e $M_j(S)$ rispetto alla media $\overline{M}(S)$ siano di 1 e 19 rispettivamente: ciò significa che la partizione i contiene un numero di intervalli molto vicino alla media, mentre la partizione j si discosta notevolmente dal valor medio. Si supponga ora che con una diversa scelta S' dei bit indice gli scostamenti di $M_i(S)$ e $M_j(S)$ diventino di 10 e 11 rispettivamente: il numero di intervalli risulta meglio bilanciato in questo secondo caso, ma il secondo addendo della (1) fornisce un punteggio migliore nel primo caso (20 contro 21) imponendo così la scelta sfavorevole dell'insieme di bit S'. Per risolvere questo problema gli autori sostituiscono la somma presente nella (1) con una somma di tipo quadratico, ottenendo così la nuova funzione di costo

$$\mathrm{Score2}(S) = \sum_{i=1}^{2^{|S|}} M_i(S) + w \left(\frac{1}{2^{|S|}} \sum_{i=1}^{2^{|S|}} (\overline{M}(S) - M_i(S))^2 \right)^{1/2}. \qquad (2)$$

Inoltre, per migliorare ulteriormente il comportamento della funzione risulta conveniente incrementare il valore di w. Lo studio sperimentale compiuto dagli autori suggerisce che il valore ottimale di w è compreso

tra 80 e 160: per confronto, si ricordi che nel lavoro precedente [3] si era posto $w = 1$. Il valore ottimale di w è comunque funzione dei parametri della cache, della tabella di routing che si sta utilizzando e anche delle caratteristiche del traffico presente sulla rete: ottimizzare la funzione di costo può dunque essere un'attività estremamente onerosa. Per questa ragione, gli autori propongono un secondo approccio per la riduzione dello sbilanciamento a cui viene dato il nome di *Variable Cache Sets Mapping* (VCSM).

Con VCSM gli intervalli di indirizzi associati ad una singola configurazione dei k bit indice possono essere anche ripartiti tra più cache set: in questo modo, alle configurazioni di bit cui corrisponde un maggior numero di partizioni viene assegnato un maggior numero di cache set, annullando così lo sbilanciamento. La ripartizione viene effettuata utilizzando n bit indice aggiuntivi. Più precisamente, la ripartizione richiede innanzitutto la determinazione dell'insieme S di k bit secondo l'algoritmo visto per la strategia IHARC; dei $32 - k$ bit rimasti, gli n bit più significativi vengono poi utilizzati per assegnare gli intervalli di indirizzi ai cache set: a ciascuna configurazione dei $k + n$ bit or ora specificati può dunque corrispondere un diverso cache set. L'assegnazione avviene sulla base di una coppia (i, j) di potenze di 2 e secondo la procedura che segue.

1. Le 2^k partizioni dello spazio degli indirizzi IP indotte da S vengono ordinate per numero di intervalli crescente.

2. Le prime $(ij - i)/(ij - 1)2^k$ partizioni della sequenza ordinata vengono assegnate ai primi $(j - 1)/(ij - 1)2^k$ cache set in gruppi di i partizioni consecutive.

3. Ciascuna partizione non ancora assegnata viene suddivisa tra j cache set.

La scelta dei parametri i e j è ancora una volta affidata a considerazioni euristiche o al risultato di prove sperimentali: in [8] la coppia che risulta fornire migliori prestazioni è $(1024, 8)$. Il vantaggio di VCSM è comunque quello di non richiedere lunghe sessioni di prova per ottimizzare il comportamento della cache: il bilanciamento degli intervalli di indirizzi IP è infatti assicurato indipendentemente dalla bontà della funzione di costo adottata. Bisogna tuttavia sottolineare che VCSM richiede supporto hardware supplementare per gestire gli n bit indice aggiuntivi, e che i risultati sperimentali mostrano che VCSM offre prestazioni inferiori rispetto a quelle ottenibili con la funzione (2) opportunamente ottimizzata[‡].

Un ulteriore punto debole della strategia IHARC è, come si è già accennato, legato all'aggiornamento della tabella di routing: ad ogni aggiornamento è infatti necessario invalidare il contenuto della cache ed è inoltre richiesto il ricalcolo dei bit indice. A questo proposito, in [8] si osserva che l'invalidazione della cache è inevitabile se si vuole evitare di commettere errori nel routing, ma che un aggiornamento della tabella influenza solo una porzione della cache: più precisamente detto m il numero di bit significativi di un netmask che non appartengono all'insieme S dei bit indice si ha che tale netmask coinvolge al più 2^m delle partizioni individuate da S. È dunque sufficiente invalidare solo le linee di cache associate a queste 2^m partizioni, limitando così al minimo la perdita di efficienza. Contrariamente all'invalidazione, il ricalcolo dei bit indice non è indispensabile se si accetta un lieve decadimento delle prestazioni: in effetti, gli esperimenti effettuati dagli autori mostrano che i bit indice possono essere ricalcolati ogni ben 25000 aggiornamenti della tabella di routing con un incremento del tempo medio d'accesso limitato a qualche decina di picosecondi.

Riferimenti bibliografici

[1] X. Chen. Effect of caching on routing-table lookup in multimedia environments. In *IEEE INFOCOM*, pages 1228–1236, April 1991.

[2] Tzi-Cker Chiueh and Prashant Pradhan. High performance IP routing table lookup using CPU caching. In *IEEE INFOCOM*, April 1999.

[‡] Si noti che i risultati di [3] e di [8] non sono confrontabili in quanto è diverso il set-up degli esperimenti.

[3] Tzi-Cker Chiueh and Prashant Pradhan. Cache memory design for network processors. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, pages 409–418, 2000.

[4] R. de la Briandais. File searching using variable-length keys. In *Proceedings of the AFIPS Western Joint Computer Conference*, pages 295–298, San Francisco, USA, March 1959.

[5] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *Proceedings of ACM SIGCOMM 97*, pages 3–14, Cannes, France, September 1997.

[6] O. Feldmeier. Improving gateway performance with a routing-table cache. In *IEEE INFOCOM*, pages 298–307, March 1988.

[7] Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, September 1960.

[8] Kartik Gopalan and Tzi-Cker Chiueh. Improving route lookup performance using network processor cache. In *SC2002 High Performance Networking and Computing*, Baltimore, MD, November 2002.

[9] Neeraj Gulati, Carey L. Williamson, and Richard B. Bunt. Lan traffic locality: Characterization and application. In *Local Area Network Interconnection. Proceedings of the First International Conference*, pages 233–250, October 1993.

[10] R. Jain. Characteristics of destination address locality in computer networks: a comparison of caching schemes. *Computer Networks and ISDN Systems*, 18(4):243–254, May 1990.

[11] R. Jain and S. Routhier. Packet trains: Measurements and a new model for computer network traffic. *IEEE Journal of Selected Areas in Communications*, SAC-4(6):986–995, September 1986.

[12] Jeffrey C. Moghul. Network locality at the scale of processes. *ACM Transactions on Computer Systems*, 10(2):81–109, May 1992.

[13] Craig Partridge. Locality and route caches. In *NSF Workshop on Internet Statistics Measurement and Analysis*, 1996.

[14] Bryan Talbot, Timothy Sherwood, and Bill Lin. IP caching for terabit speed routers. In *Global Communications Conference (Globecom'99)*, pages 1565–1569, Rio de Janeiro, Brazil, December 1999.

[15] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high-speed prefix matching. *ACM Transactions on Computer Systems*, 19(4):440–482, November 2001.

[16] Jun Xu, Mukesh Singhal, and Joanne Degroat. A novel cache architecture to support layer-four packet classification at memory access speeds. In *IEEE INFOCOMM*, pages 1445–1454, Tel Aviv, Israel, March 2000.