1

Misura del traffico

Esercizio 1.1 Si consideri un sistema di misura del traffico in un router. Ai pacchetti entranti da tutti i link è associato un identificativo di flusso x ottenuto concatenando alcuni campi del pacchetto. La lunghezza dei pacchetti è costante di 512 byte.

Sull'identificativo x sono calcolati n hash usando la famiglia di funzioni indipendenti $h_i(x)$ con $i = 1, \ldots, k$. Tali hash sono usati come indici in n array $A_1, \ldots, A_n$ di m elementi ciascuno. Ad ogni pacchetto arrivato si incrementa di uno la $h_n(x)$-esima posizione di ognuno degli n array.

Se $A_n[h_n(x)] \geq S$ per ogni n il valore x è salvato su disco.

Tutti gli array sono azzerati ogni $T = 10\,\text{s}$.

a) Dire quali campi occorre scegliere per rappresentare l'identificativo del flusso per avere nello stesso flusso tutti i pacchetti diretti allo stesso socket sullo stesso host.

b) Considerando un link unidirezionale a $C = 40\,\text{Gbit/s}$, dire quanti bit occorrono per rappresentare gli n array.

c) Dire come scegliere la soglia S in modo che si scriva su disco tutti i flussi da almeno 25 Mbit/s.

d) Un certo flusso y ha superato la soglia nell'attuale intervallo di misura. Considerando $m = 2^8$ e $n = 3$, dire qual è la probabilità che un nuovo flusso sia copiato su disco. *Ignorare gli eventuali altri flussi nel sistema.*

e) Calcolare lo spazio occupato e la probabilità di falso positivo nei due casi $\{m = 2^4, n = 4\}$ e $\{m = 2^8, n = 2\}$. Spiegare vantaggi e controindicazioni di ciascuna opzione.

Soluzione

a)
$$x = \text{IPd} \| \text{proto} \| \text{portd}$$

b) Nel caso pessimo, in 10 s si accumulano $400 \cdot 10^9$ bit in un unico contatore. Contando in pacchetti servono
$$\left\lceil \log_2 \frac{400 \cdot 10^9}{8 \times 512} \right\rceil mn = 27mn \text{ pacchetti}$$
(oppure $39mn$ bit o $36mn$ ottetti)

c) In dieci secondi il flusso da monitorare accumula almeno 250 Mbit, ovvero $S = 61036$ pacchetti. (oppure $31 \cdot 10^7$ ottetti)

d)
$$\left(\frac{1}{m}\right)^n = \frac{1}{2^{24}} \simeq 6 \cdot 10^{-8}$$

e) La probabilità di falso positivo è identica $2^{-16} = 1{,}5 \cdot 10^{-5}$. Lo spazio occupato è $27 \times 2^6 = 1{,}7$ kbit nel primo caso e $27 \times 2^9 = 14$ kbit nel secondo caso. Nel secondo caso però si devono calcolare due hash, anziché 4.

1.1 IP Traceback

Dato un pacchetto P, registrato in un log di un host o di un Intrusion Detection System V, si vuole ricostruire la lista dei nodi attraversati per giungere a destinazione. Nota che il routing è spesso asimmetrico, quindi non è corretto semplicemente inviare un pacchetto di risposta e tracciarne il percorso.

La strategia di *IP traceback via Logging* consiste nel salvare in ogni nodo un log di tutti i pacchetti transitati dal nodo.

Una volta noto P in V si interrogano i vicini di V chiedendo se nel log è presente P. Normalmente dovrebbe rispondere positivamente un solo vicino, R_1. Si procede ricorsivamente fino al bordo della rete.

L'approccio qui descritto è impraticabile per vari motivi:

- i pacchetti possono cambiare lungo la rete

- i requisiti di memoria e di velocità di accesso sono proibitivi
- i log stessi sono un obiettivo per intrusi

È preferibile una soluzione basata su hash e «filtri di Bloom».

Un filtro di Bloom è una struttura efficiente per implementare il test di appartenenza a un insieme.

Siano:

- $h_i(\cdot)$ una famiglia di k funzioni hash con output tra 0 e $m-1$.
- A un array binario di m elementi, inizialmente nullo

Sono definite le seguenti operazioni.

procedure INSERT(x)
  for $i \leftarrow 1, k$ **do** ▷ for all the hash functions
    $A[h_i(x)] = 1$ ▷ set to 1 the array at index $h_i(x)$
  end for
end procedure
procedure TEST(x) ▷ Tests if the element x is in the set
  for $i \leftarrow 1, k$ **do** ▷ for all the hash functions
    if $A[h_i(x)] == 0$ **then**
      return 0 ▷ x is not in the set
    end if
    return 1 ▷ x is (probably) in the set
  end for
end procedure

I tempi di inserimento e di test sono costanti, ma il test può dare luogo a falsi positivi. Se n è il numero di elementi (distinti) nel set, allora la probabilità di falso positivo è:

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \simeq \left(1 - \exp\left(-\frac{kn}{m}\right)\right)^k$$

È possibile implementare l'algoritmo di IP traceback creando dei log sintetici usando i filtri di Bloom. Sia x una stringa di bit ottenuta concatenando alcuni campi del pacchetto P che non cambiano da nodo a nodo. In ogni nodo si effettua un inserimento nel filtro.

Poiché la probabilità di falso positivo cresce nel tempo, quando essa diventa troppo grande si salva il filtro e lo si resetta.

Esercizio 1.2 Si consideri un filtro di Bloom costituito da un array di m bit e k funzioni hash. Inizialmente tutti i bit dell'array sono posti a zero. Si assuma che ciascuna posizione all'interno dell'array può essere selezionata da ciascuna funzione di hash con la medesima probabilità.

a) Calcolare la probabilità che un dato bit sia posto a 0 dopo aver effettuato l'inserimento di 1 elemento.

b) Calcolare la probabilità che un dato bit sia posto ad 1 dopo aver effettuato l'inserimento di n elementi.

c) Calcolare la probabilità di un falso positivo dopo aver effettuato l'inserimento di n elementi.

d) Il valore di k che minimizza la probabilità di falso positivo è $\frac{m}{n} \log 2$. Calcolare la probabilità ottima di falso positivo.

e) Calcolare la grandezza del filtro m per una probabilità di flaso positivo ϵ.

Soluzione

a)
$$\left(1 - \frac{1}{m}\right)^k$$

b)
$$1 - \left(1 - \frac{1}{m}\right)^{kn}$$

c)
$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

d)
$$\left(\frac{1}{2}\right)^{m/n \log 2} = 0.6185^{m/n}$$

e)
$$0.6185^{m/n} = \epsilon$$
$$\frac{m}{n} \log_2(0.6185)) = \log_2 \epsilon$$
$$m = 1.44n \log_2(1/\epsilon)$$

quindi $1.44 \log_2(1/\epsilon)$ bit per ogni elemento atteso.

Esercizio 1.3 Si consideri un filtro di Bloom con $m = 5$ e $k = 2$. Inoltre

- $h_1(x) = x \bmod 5$
- $h_2(x) = (2x + 3) \bmod 5$

Inizialmente tutti i bit dell'array sono posti a zero.

a) Inserire 9 e 11 nel filtro di Bloom. Mostrare lo stato di ogni bit prima e dopo ogni inserimento.

b) Verificare l'appartenza al dataset degli elementi 15 e 16 e commentare i risultati ottenuti (vi sono o meno falsi positivi?)

Soluzione Situazione iniziale:

0	0	0	0	0

Inserimento di 9. Si ha $h_1(9) = 4$ e $h_2(9) = 1$.

0	1	0	0	1

Inserimento di 11. Si ha $h_1(9) = 1$ e $h_2(9) = 0$.

1	1	0	0	1

Verifica la presenza di 15. Si ha $h_1(15) = 0$ e $h_2(15) = 3$.

1	1	0	0	1

Verifica la presenza di 16. Si ha $h_1(16) = 1$ e $h_2(16) = 0$.

1	1	0	0	1

Quindi $x = 16$ dà luogo a un falso positivo.

Esercizio 1.4 Si consideri un filtro di Bloom costituito da un array di 16 bit e 3 funzioni hash. Inizialmente tutti i bit dell'array sono posti a zero.

a) Calcolare la probabilità che un dato bit sia posto ad 1 dopo aver effettuato l'inserimento di 3 elementi

Soluzione Sia m il numero di bits del filtro, k il numero di funzioni di hash e n il numero di inserimenti. La probabilità si ottiene come:

$$1 - \left(1 - \frac{1}{m}\right)^{kn} = 1 - \left(1 - \frac{1}{16}\right)^{9} \approx 0.44$$

b) Calcolare la probabilità di un falso positivo.

Soluzione La probabilità di un falso positivo si ottiene come:

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k}=\left(1-\left(1-\frac{1}{16}\right)^{9}\right)^{3}\approx 0.085$$

c) Dire se la funzione $h(x) = 4x \bmod 16$ è una buona funzione di hash per questo filtro.

Soluzione No, perché gli ultimi due bit sono sempre 0.

d) Sono date le funzioni hash

$$h_1(x) = 3x \bmod 16$$
$$h_2(x) = 5x \bmod 16$$
$$h_3(x) = 7x \bmod 16$$

Inizialmente tutti i bit dell'array sono posti a zero. Inserire $x = 1$ e $x = 4$ nel filtro di Bloom. Mostrare lo stato del filtro dopo ogni inserimento.

Soluzione

Iniziale	0000000000000000
$x = 1$	0001010100000000
$x = 4$	0001110100001000

e) Conclusi gli inserimenti, verificare l'appartenza al dataset degli elementi $x = 5$ e $x = 2$. Dire se ci sono falsi positivi.

Soluzione $0001000001000001 \subset 0001110100001000$
$0000001000100010 \not\subset 0001110100001000$

Esercizio 1.5 Per aumentare la capacità di un sistema Gigabit Ethernet si usano 8 cavi in parallelo. La lunghezza media di un pacchetto è 500 byte. La scelta del cavo da usare è effettuata calcolando un hash $h(\cdot)$ su alcuni campi del pacchetto. Gli ultimi 3 bit dell'hash sono usati per scegliere il cavo da usare per il pacchetto.

a) Dire quali campi del pacchetto inserire nel hash per garantire che una singola connessione TCP usi uniformemente tutti i cavi in parallelo. Si supponga che 8 pacchetti si presentino al sistema. Qual è la probabilità che ogni pacchetto usi un cavo diverso?

Soluzione Occorrono campi includano campi diversi da pacchetto a pacchetto, per esempio: ID pacchetto, sequence number, payload. Un possibile soluzione è hash di tutto il pacchetto.

Delle 8^8 possibili combinazioni, quelle in cui i valori sono tutti diversi sono le permutazioni di 8 elementi: 8!. Per cui la probabilità è $\frac{8!}{8^8} \simeq 0.0024$

b) Dire quali campi del pacchetto inserire nel hash per garantire che una singola connessione TCP usi un solo cavo in entrambe le direzioni.

Soluzione Bisogna usare i 5 campi che identificano il socket: IP sorgente/destinazione, protocollo, porta sorgente/destinazione. Per garantire che l'hash sia uguale nelle due direzioni si inseriscono gli indirizzi IP e i numeri di porta in ordine di valore.

$$h\Big(\min(\mathsf{IPsrc},\mathsf{IPdst})\|\max(\mathsf{IPsrc},\mathsf{IPdst})\|\mathsf{protocollo}\| \\ \|\min(\mathsf{porta\ src},\mathsf{porta\ dst})\|\max(\mathsf{porta\ src},\mathsf{porta\ dst})\Big)$$

c) Definire un meccanismo che riservi uno dei cavi al solo traffico UDP.

Soluzione I pacchetti UDP si mandano al cavo numero 7. Per tutti gli altri pacchetti, anziché usare selezionare il cavo usando h(pacchetto) mod 8, si seleziona il cavo usando h(pacchetto) mod 7.

d) Definire un meccanismo che permetta di campionare approssimativamente un pacchetto al secondo su ogni cavo. Poiché la rete è composta da molti nodi, il meccanismo deve campionare gli stessi pacchetti in tutti i nodi.

Soluzione Arrivano circa $10^9/(500 \cdot 8) = 250000$ pacchetti al secondo. Supponiamo di usare una funzione hash a 32 bit. Gli ultimi tre bit si scartano, perché per ogni cavo sono fissati. Rimangono 29 bit che sono interpretati come un numero da 0 a 2^{29}. Se tale numero è minore di $2^{29}/250000 \simeq 2148$ si campiona il pacchetto.

1.2 Misura del traffico in ryu

Lo switch openflow raccoglie statistiche per flusso e per porta.

Le statistiche per flusso sono consegnate al controllor quando il flusso termina (evento FlowRemoved) oppure in risposta alla richiesta del controller (evento FlowStatsReply). Le statistiche per porta sono consegnate al controllor risposta alla richiesta del controller (evento PortStatsReply).

L'uso tipico consiste nel inviare periodicamente delle richieste (FlowStatsRequest e PortStatsRequest) non bloccanti e gestire le risposte nel modo usuale, ovvero con una funzione che gestisce gli eventi FlowStatsReply e PortStatsReply.

In ryu è possibile usare dei greenthread, ovvero delle coroutine che sono eseguite tra un evento e il successivo.

Per lanciare un greenthread si usa la funzione `hub.spawn` fornita dalla libreria ryu. Per sospendere un greenthread si usa la funzione `hub.sleep`.

switch-monitor.py

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER,
↪  MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet
from ryu.lib import hub

# This implements a learning switch in the controller
# The switch sends all packet to the controller
# The controller implements the MAC table using a python dictionary
# The controller prints flow statistics

# eseguire con:
# ryu-manager --verbose

class PsrSwitch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(PsrSwitch, self).__init__(*args, **kwargs)

        # tabella dei MAC
        self.mac_to_port = {}
```

```
        # tabella dei datapath
        self.datapaths = {}

        # thread che lancia periodicamente le richieste
        self.monitor_thread = hub.spawn(self._monitor)

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

    def _request_stats(self, datapath):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)
        req = parser.OFPPortStatsRequest(datapath, 0,
        ↪  ofproto.OFPP_ANY)
        datapath.send_msg(req)

    # execute at switch registration
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures,
    ↪  CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        self.datapaths[datapath.id] = datapath
        self.mac_to_port.setdefault(datapath.id, {})

        # match all packets
        match = parser.OFPMatch()
        # send to controller
        actions = [
            parser.OFPActionOutput(
                ofproto.OFPP_CONTROLLER,
                128
            )
        ]
```

```
        inst = [
            parser.OFPInstructionActions(
                ofproto.OFPIT_APPLY_ACTIONS,
                actions
            )
        ]
        mod = parser.OFPFlowMod(
            datapath=datapath,
            priority=0,
            match=match,
            instructions=inst
        )
        datapath.send_msg(mod)


    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        in_port = msg.match['in_port']
        dpid = datapath.id

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        assert eth is not None

        dst = eth.dst
        src = eth.src

        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

#        self.logger.info("packet in %s %s %s %s %s", dpid, src,
↪  dst, in_port, out_port)
```

```
        actions = [
            parser.OFPActionOutput(out_port)
        ]

        assert msg.buffer_id != ofproto.OFP_NO_BUFFER

        if out_port != ofproto.OFPP_FLOOD:
          # install a flow and send the packet
            match = parser.OFPMatch(
                eth_src=src,
                eth_dst=dst
            )
            inst = [
                parser.OFPInstructionActions(
                    ofproto.OFPIT_APPLY_ACTIONS,
                    actions
                )
            ]
            ofmsg = parser.OFPFlowMod(
                datapath=datapath,
                priority=1,
                match=match,
                instructions=inst,
                buffer_id=msg.buffer_id
            )
        else:
            # only send the packet
            ofmsg = parser.OFPPacketOut(
                datapath=datapath,
                buffer_id=msg.buffer_id,
                in_port=in_port,
                actions=actions,
                data=None
            )

        datapath.send_msg(ofmsg)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body
        self.logger.info('datapath         '
        'match        '
```

```
            'out-port packets bytes')
        self.logger.info('---------------- '
        '-------- ----------------- '
        '-------- -------- --------')
        for stat in body:
            self.logger.info('%016x %26s %8x %8d %8d',
                ev.msg.datapath.id,
                stat.match,
                stat.instructions[0].actions[0].port,
                stat.packet_count, stat.byte_count)

    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body
        self.logger.info('datapath port '
        'rx-pkts rx-bytes rx-error '
        'tx-pkts tx-bytes tx-error')
        self.logger.info('---------------- -------- '
        '-------- -------- -------- '
        '-------- -------- --------')
        for stat in body:
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                ev.msg.datapath.id, stat.port_no,
                stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```