INDIRIZZI MAC E COLLEGAMENTO DI UN PC ALLA RETE

Un **indirizzo MAC** è composto da **48 bit** e suddiviso in 6 ottetti: i **primi 24 bit** identificano il **costruttore della scheda di rete**, i **restanti** identificano la **scheda di rete** stessa e sono decisi dal costruttore, il quale ha a disposizione circa 16 milioni (2^24) indirizzi diversi. Un **indirizzo MAC identifica univocamente un device all'interno della rete**, può accadere che in una rete ci siano indirizzi duplicati, ma solitamente è compito del costruttore smistare indirizzi uguali in reti diverse.

Ethernet è la **tecnologia di trasmissione utilizzata al livello datalink** dello stack TCP/IP. Si tratta di un **cavo seriale composto da 8 fili**, i quali sono suddivisi in 4 coppie: **ricezione e trasmissione sono separate**, così da evitare collisioni tra i pacchetti. La **quantità minima** trasmissibile è di **60 bytes** e la **velocità di trasmissione** odierna è di circa **100 Gbps**.

Per connettere uno o più device ad una rete, in origine si utilizzava un hub, ossia un ripetitore che inoltrava un pacchetto su tutte le sue porte. Si avevano però problemi legati all'occupazione della banda e i pacchetti venivano inviati a tutti i device della rete. Così con gli anni si è passati ad un'evoluzione dell'hub, lo **switch**, che **opera a livello datalink effettuando un filtraggio sugli indirizzi MAC** ed inviando i pacchetti solo alla destinazione specificata. Per far questo, lo switch si tiene una **tabella contenente le associazioni <porta,indirizzo MAC>, detta tabella ARP**. Per collegare un device ad una rete gli step eseguiti sono:

1- **si collega il device** alla rete **tramite il cavo**
2- con il protocollo ARP **si invia in broadcast un pacchetto**, il quale **viene ricevuto dallo switch, che va ad aggiornare la sua tabella ARP** con le associazioni
3- se ci sono **due device con stesso indirizzo MAC** all'interno della stessa rete si distinguono **due casi**

   3.1- **lo switch fa connettere alla rete il device con indirizzo MAC già presente**. In questo modo **entrambi i dispositivi riceveranno gli stessi pacchetti**. Questa soluzione è adottata da apparati di rete più economici

   3.2- **lo switch butta giù una delle due porte o entrambe**. Questa soluzione, adottata da dispositivi di rete più costosi e sicuri, **permette di evitare la nascita di un loop di rete** tra le due porte con stesso indirizzo MAC. Infatti, potrebbe nascere una **situazione in cui** ad esempio in uno switch a 4 porte **ho due device collegati su due porte diverse** (es. 1 e 4) **con stesso indirizzo MAC**. Se porta1 invia **un pacchetto** a porta4, una volta giunto a destinazione, il pacchetto avrà come destinazione porta1, quindi **rimbalzerà all'infinito tra le due porte.**

Ogni porta dello switch ha un indirizzo MAC che la identifica. Un **indirizzo MAC** all'interno della rete **potrà cambiare**. Al suo avvio, un device prende l'indirizzo MAC della propria scheda di rete, ma a livello software tale indirizzo può cambiare. All'interno del pacchetto, gli indirizzi IP sorgente e destinazione restano immutati (a meno di router NAT), mentre **gli indirizzi MAC cambiano ad ogni passo**, poiché la **comunicazione** è **punto-punto**.

MONITORAGGIO DEL TRAFFICO DI RETE

In genere tutto il traffico di rete deve essere catturato, non solo il traffico IP o solo quello verso l'**host sniffer**. Questo host dovrà **avere i permessi di root o delle apposite capabilities**, per vedere i pacchetti di tutti i processi in esecuzione, e dovrà stare all'interno di una VM (virtualizzazione del SO con un proprio kernel) o di un container (condivide il kernel con la macchina fisica), in modo da intercettare anche il traffico sulla loopback. Esistono diverse **tecniche di monitoraggio, suddivise in hardware e software**. A livello software si distingue:

- **switch port mirror**: serve per comandare allo switch di inoltrare il traffico da e verso un host su una specifica porta dello sniffer
- **switch vlan mirror**: una vlan serve per separare traffici diversi e non mescolare i dati. Gli host staranno sullo stesso cavo condiviso senza saperlo

Uno dei problemi maggiori è dato dal fatto che **sullo sniffer si utilizza solo la parte di ricezione del cavo Ethernet**, mentre **sull'host che sto monitorando posso**, in teoria, **inviare e ricevere dati** in contemporanea. Infatti, se ho un cavo Ethernet da 1 Gb, significa che ogni singolo cavetto che lo compone è utilizzabile per 1 Gb. Nella situazione di prima quindi, l'host monitorato può arrivare a generare 2 Gb di traffico, mentre lo sniffer ha capacità di 1 Gb, poiché utilizza solo il cavo di ricezione. **Può accadere che lo sniffer potrebbe perdersi del traffico**, quindi **per evitare ciò si potrebbe dotare lo sniffer di una scheda di rete più potente** (es. 10 Gb) **ed abbinarci** magari **l'utilizzo di alcune porte dedicate dello switch**, le quali consentono una **velocità di trasferimento dati maggiore** rispetto alle altre.

Un **dispositivo hardware per il monitoraggio del traffico di rete** è il **network tap**: si tratta di un **device** fisico **che**, se **attaccato ad una porta dello switch, la duplica**. Il tap **si occupa** anche di **separare il traffico in entrata ed in uscita da quella porta**, quindi potrebbe risultare più complicato ricostruire le interazioni della comunicazione, ma di contro ho sempre mittente e destinatario fissati.

GESTIONE DELLA RETE

Un **managed object** è **un'astrazione di una risorsa reale** e le sue principali componenti sono:

- **Attributi**: descrivono lo **stato dell'oggetto**, possono cambiare ed essere manipolati
- **Operazioni**: rendono possibile **l'accesso al managed object**
- **Comportamento**: semantica ed interazione con al risorsa reale
- **Notifiche**: **messaggi** generati dal managed object **in seguito a specifici eventi**

Un **MIB** (Management Information Base) rappresenta l'**unione di tutti i managed** objects ed il suo utilizzo **consente di modularizzare la gestione del sistema**. Un **paradigma** utilizzato nella gestione di rete è **Agent-Manager**: l'**agent** sta **in relazione diretta con la risorsa**, **implementa il MIB accedendo alla risorsa reale**, **riceve richieste dal manager** e invia risposte e **notifica i cambiamenti di stato** della risorsa; il **manager esercita il controllo**, **gestisce le operazioni** di management attraverso gli appositi protocolli ed **elabora le risposte** ricevute dall'agent.

Solitamente, **agent e manager stanno su host differenti** e le applicazioni per il management in generale devono coprire queste 5 aree (FCAPS):

- **Fault:** gli errori devono essere identificati, isolati, notificati e risolti
- **Configurazione:** informazioni sulla configurazione fisica del sistema e sulle specifiche
- **Account:** informazioni sul consumo e sull'utilizzo delle risorse
- **Performance:** misure sull'accesso alle risorse e sull'efficienza, sia lato utente che lato fornitore
- **Sicurezza:** politiche di sicurezza, generazione di account e passwords, etc…

Un **servizio** è una **funzionalità astratta fornita dalla rete**, e ci si accede **tramite un'interfaccia** detta **SAP** (Service Access Point). I servizi **si suddividono in confermati** (richiesta-risposta) **e non confermati** (solo richiesta).

ABSTRACT SYNTAX NOTATION ONE (ASN.1)

Dato che agent e manager risiedono su host differenti, negli anni è stato sollevato il **problema** di compatibilità **riguardo alla sintassi di comunicazione**. Generalmente, la sintassi utilizzata per trasferire informazioni sul web è formata dalle stringhe, il che rende la cosa inefficiente. **ASN.1** è una **sintassi usata per dare un formato standard ai dati**, alle strutture e ai messaggi. L'**obiettivo** di ASN.1 è quello di **consentire lo scambio di informazioni tra macchine con architetture diverse**, rendendo i loro messaggi indipendenti dai linguaggi di programmazione. ASN.1 consente di **definire la sintassi in modo dinamico e le strutture dati durante il trasferimento,** detto anche fase di negoziazione. Il **BER** (Basic Encoding Rules) determina come un tipo di dato di ASN.1 può essere rappresentato in stringa, **consente di passare da un formato astratto ad un formato on wire** ed è **basato sull'algoritmo TLV** (Tag-Length-Value). La codifica tramite questo algoritmo **consente al destinatario di ricostruire il messaggio ricevuto.**

Un **esempio** riguarda il **problema del big endian e little endian**. L'endian si riferisce a come una data macchina memorizza i bytes: nel big endian si memorizza a partire dal bit più significativo, mentre nel little endian si parte dal bit meno significativo. È **molto importante** quindi **stabilire una sintassi comune tra due macchine diverse**, per evitare di ricevere dati invertiti ed errati.

ASN.1 ha anche dei tipi tra cui booleani, interi, bitstring, octetstring, enumerated e **object identifier**. Un object identifier (OID) **serve per identificare univocamente un nodo dell'ISO Registration Tree**. Esso è un albero, in continua espansione, e si usa per identificare univocamente un oggetto. **Ogni oggetto è identificato da un path dalla radice al nodo stesso**, e tale path è denotato **tramite una sequenza di interi**, appunto l'OID.

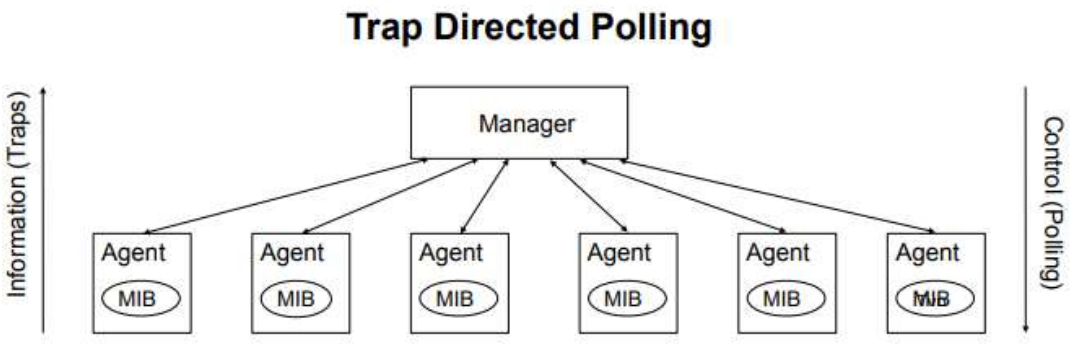
INTERNET MANAGEMENT E SNMP

Gestire una rete significa identificare i vari problemi, isolarli e risolverli. Per la gestione di una rete si rende necessario avere un insieme di protocolli e regole tale da mantenere attiva e consistente l'intera struttura nel tempo. Il **protocollo standard per gestire una rete ed i suoi dispositivi** è **SNMP** (Simple Network Management Protocol) ed i suoi obiettivi sono:

- **Costi di sviluppo bassi:** si tende a **ridurre i costi e la complessità delle funzionalità** per la gestione, utilizzando le stesse tecnologie per ogni device della rete
- **Estensibilità:** si vuole progettare **cose durevoli nel tempo** tramite la definizione di nuovi MIB
- **Indipendenza:** si vuole **indipendenza rispetto all'architettura e alla macchina fisica**
- **Robustezza:** si vuole utilizzare **protocolli di comunicazione semplici**, come **UDP**. Esso è un protocollo **adatto per devices piccoli e dotati di poca memoria**, ed è più robusto di TCP, poiché non si devono gestire connessioni durature, non si devono mantenere una tabella di connessioni attive o altre struttre dati ausiliarie e non è esposto ad attacchi come DDoS

SNMP è un **protocollo trasparente alla comunicazione e ai dati**, non interferisce. La **suddivisione tra la parte di gestione e quella dei dati** si realizza **attraverso le vlan o utilizzando cavi fisici diversi**. Negli anni sono state sviluppate 3 versioni diverse. Per quanto riguarda il paradigma agent-manager, SNMP dovrà garantire l'ubiquità, ossia devices diversi verranno gestiti allo stesso modo indipendentemente dall'architettura della rete. Il **funzionamento di SNMP** si basa principalmente su **due meccanismi**:

- **Polling: meccanismo di controllo**, si configurano gli agent del sistema, dopodichè ad intervalli di tempo regolari il **manager contatterà gli agents** per verificare il loro corretto funzionamento ed il loro stato. Le **informazioni richieste** sono **generiche**, si vuole conoscere solamente se il device è attivo, da quanto tempo e poco altro. L'**intervallo di tempo** che passa tra un polling ed il successivo **dipende dal grado di fiducia che il manager ha verso l'agent**, quindi più un agent da problemi più verrà controllato frequentemente
- **Trap: meccanismo di informazione**, un **agent informa il manager** a seguito di eventi anomali che hanno causato un cambio di stato (es. interruzione di un servizio)



SNMP ha ereditato alcune parti di ASN.1, come i tipi, in più sono stati **definiti altri tipi di dato** come integer, unsigned, **gauge** (oggetto con valore compreso in un range [min,max]) e **counter** (oggetto che deve essere interpretato in base al contesto, il valore è la differenza tra due misurazioni consecutive). Inoltre, in SNMP non si utilizzano strutture dati complesse ma si lavora con:

- **Scalari:** esistono una sola volta per ogni device (es. nome host)
- **Informazioni tabellari:** possono esserci più volte e sono inserite in una tabella (es. interfacce di rete). Un ruolo importante nelle tabelle è svolto dalla colonna "index", definita nel MIB, contenente valori tutti diversi e con il compito di riferire univocamente un oggetto della tabella. Per accedere ad un **elemento di una tabella**, dobbiamo specificare **l'OID della tabella + indice di colonna dove sta l'oggetto + indice di riga**

Le uniche operazioni definite sono read e write, dopodichè agent e manager risiederanno su host diversi ed ognuno avrà il proprio indirizzo IP. Il **manager dovrà conoscere l'IP dell'agent**, mentre esso sfrutterà la porta nota 161 per ricevere le richieste. Un agent sarà composto da più oggetti diversi, ciascuno denotato da un OID, in più **all'interno dello stesso agent ci potranno essere più istanze di uno stesso oggetto**: per questo **si usa anche l'Istance Identifier**, il quale viene concatenato all'OID.

Per poter accedere alle tabelle e navigare all'interno dell'ISO Registration Tree si ha bisogno di algoritmi efficienti, per questo è stata implementata una **visita con ordine lessicografico**. Si tratta di una **visita in profondità effettuata sull'albero**, utile per definire al suo interno il concetto di "successore".

Le **informazioni scambiate tra agent e manager si mantengono nel MIB**, rappresentato come un file testuale con alcune macro:

- **Module identity**: **prima parte**, **unica, per ogni MIB**, da informazioni sulla versione Ethernet in uso, la tipologia di oggetti che compongono l'agent e la versione del MIB in uso. Per garantire la compatibilità tra MIB di versioni diverse, gli oggetti non vengono eliminati ma marcati con dei tag appositi.
- **Object identity**: Gli **oggetti intermedi** per percorrere l'ISO Registration Tree **si definiscono con questa macro,** contenente uno stato ed una descrizione per gli utenti.
- **Object type**: Dopo aver creato **un oggetto**, esso **viene definito con questa macro** ed un nome simbolico con iniziale minuscola. Ogni oggetto ha una **sintassi**, uno **stato**, una **descrizione** ed un **insieme di operazioni consentite**
- **Notification type**: Per le **notifiche** di cambio di stato si utilizza questa **macro.** L'invio della notifica avviene tramite UDP per motivi di semplicità e robustezza. In questa macro troviamo **uno o più oggetti necessari al manager** per fare una diagnostica, lo **stato** ed una **descrizione in cui si specificano gli OID o la causa della notifica**

Le **operazioni** implementate **in SNMPv1** sono le seguenti:

- **GET:** viene inviata **dal manager all'agent** e prevede una risposta. Con la get **si legge il valore di una o più variabili.** Gli errori possibili sono "too big", "no such name" o "general error"
- **GET_NEXT:** viene inviata **dal manager all'agent** e prevede una risposta. **Restituisce il valore del prossimo Istance Identifier rispetto all'ordine lessicografico**. Scorre il MIB lessicograficamente e si può utilizzare per fare più richieste consecutive. Gli errori possibili sono "too big", "no such name" o "general error"
- **SET:** viene inviata **dal manager all'agent** e prevede una risposta. **Consente di scrivere o modificare il valore di una o più istanze** tra quelle consentite nel MIB. Gli errori possibili sono "too big", "no such name", "bad value" o "general error"
- **TRAP:** viene inviata **dall'agent al manager** e non prevede una risposta, quindi non potremo mai sapere se il manager ha ricevuto la trap. Per questo motivo, in genere la trap viene inviata più volte, altrimenti male che vada c'è comunque il polling. Si tratta di un **modo asincrono per informare il manager di un cambio di stato dell'agent**. Un trap storm avviene a seguito di un blackout, quando vengono riavviati tutti i device

Un pacchetto SNMP è formato dalla versione SNMP, dalla community e da un campo PDU (Protocol Data Unit): per quanto riguarda le **operazioni get, get_next e set**, sia il **campo PDU** sia il **pacchetto di risposta** sono formati dagli **stessi campi**. Le operazioni di **get_next e set** si dicono **atomiche**, perché se si eseguono passando loro più OID come argomento, è come se acquisissero una lock e la rilasciassero alla fine di tutto. Per la **trap**, il **PDU del pacchetto** contenente la richiesta è leggermente **diverso**, poiché **ci sono campi in più**, come **ad esempio l'indirizzo IP nativo dell'agent**: è importante che ci sia questo campo, poiché l'agent potrebbe trovarsi dietro un NAT e in quel caso l'IP pubblico sarebbe diverso dall'IP vero e proprio del device. Il manager quindi dovrà conoscere l'indirizzo IP nativo per poter effettuare interventi mirati a seguito di eventi che l'agent ha notificato.

MIB-II E GRUPPI

MIB-II definisce gli object types per IP, ICMP, UDP, TCP, etc… I suoi **obiettivi** consistono nel definire la **gestione dei protocolli di rete**, **informazioni base sugli errori**, pochi oggetti di controllo, **no informazioni ridondanti** e **non deve interferire con la normale attività di rete**. Il MIB-II sta alla base del monitoraggio di rete, contiene circa 170 oggetti, è diviso in gruppi ed è presente su ogni macchina.

Il gruppo "**system**" contiene i seguenti campi:

- **sysDescr:** breve descrizione del sistema
- **sysObjectID:** modello dell'agent e informazioni sul costruttore
- **sysUptime:** indica da quanto è attivo l'agent, quindi fornisce un'idea se l'agent è stato riavviato o meno
- **sysContact:** contatto del gestore dell'agent
- **sysName:** nome del device

- **sysLocation:** locazione fisica del device
- **sysServices:** bitmap che indica quali livelli dello stack TCP/IP implementa l'agent

Il gruppo "**interface**" contiene una tabella con le informazioni sulle varie interfacce di rete della macchina. I campi della tabella sono:

- **ifIndex:** valore persistente che identifica un'interfaccia di rete, non sono valori consecutivi per consentire modifiche nel tempo e far valere il principio di estensibilità
- **ifDescr:** nome dell'interfaccia
- **ifType:** tipo dell'interfaccia
- **ifMtu:** maximum transmission unit dell'interfaccia
- **ifSpeed:** velocità (effettiva o nominale) di trasmissione dell'interfaccia
- **ifPhysAddress**: indirizzo MAC della scheda di rete dello switch o del router
- **ifAdminStatus, ifOperStatus**: forniscono informazioni sulla scheda di rete e sul suo stato
- **ifLastChange**: contiene il tempo passato dall'ultimo cambio di stato dell'interfaccia
- **ifInOctet, ifOutOctet**: pacchetti in ingresso e uscita in bytes dal device
- **ifInUcastPkts, ifInNUcastPkts**: pacchetti in ingresso unicast e non unicast
- **ifOutUcastPkts, ifOutNUcastPkts**: pacchetti in uscita unicast e non unicast
- **ifInDiscards, ifInError:** pacchetti in entrata scartati o errati
- **ifOutDiscards, ifOutError:** pacchetti in uscita scartati o errati
- **ifInUnknownProtos:** restituisce il numero di pacchetti per cui non si conosce il protocollo di livello IP
- **ifOutQLen:** fornisce informazioni sulla lunghezza della coda di uscita e sulla capacità di ricezione del destinatario

Un altro MIB importante è il **Bridge MIB**, utile per **controllare gli stati e le interfacce degli switch di livello 2 e 3**. È quasi complementare al MIB-II, poiché **fornisce informazioni sugli host connessi alle porte dello switch** esaminato, **tra cui il loro indirizzo MAC**. Con l'associazione <MAC, porta> **è possibile ricostruire una sorta di topologia della rete** e conoscere la locazione fisica di un dato host

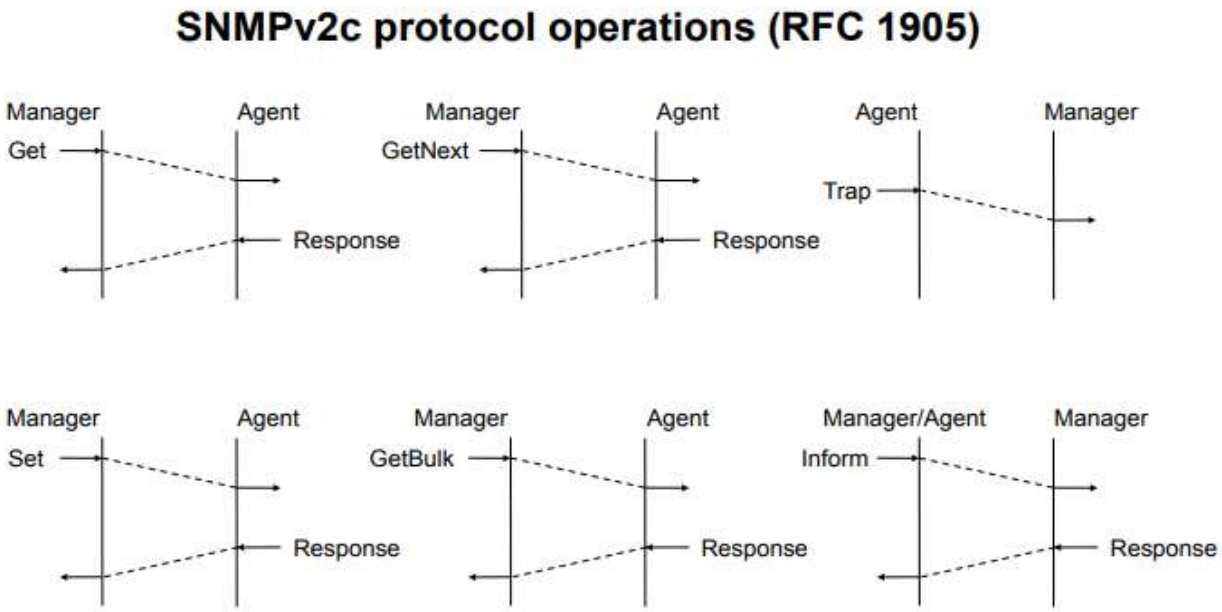
SNMPv2

La **versione 2 di SNMP**, detta appunto SNMPv2, è la più semplice versione che **ha risolto alcuni problemi di SNMPv1** e ha reso efficienti altre operazioni. Nella versione 2 sono state introdotte due **nuove primitive**, i **contatori a 64 bit**, **eccezioni più specifiche** ed espressive, tipi di dati nuovi, etc…

Una primitiva introdotta è la **GET_BULK**: si tratta di una richiesta che va **dal manager all'agent** e prevede una risposta. Serve per rendere più efficiente la get, è una sorta di **unione tra get e get_next**. Oltre a prendere come argomento degli OID, ha anche un **parametro "no repeaters"** che indica su quanti degli OID passati va fatta la get, mentre sui restanti viene fatta la get_next fino ad un limite **"max-repetitions"**. Il **vantaggio** di questa nuova primitiva è che **si riduce nettamente il**

n° di messaggi scambiati. Di **contro** però **se le richieste sono troppe, la risposta potrebbe eccedere la dimensione massima del pacchetto SNMP** e quindi non verrebbe trasmessa

L'altra primitiva introdotta è la **INFORM**, simile alla **trap** ma **confermata dal manager**. Inoltre, questa primitiva **non viene inviata esclusivamente dall'agent**, ma può accadere che in reti complesse con una gerarchia ben delineata ci siano più manager a vari livelli, quindi **può anche essere che la invii un manager ad un altro di livello superiore**.



SNMPv3

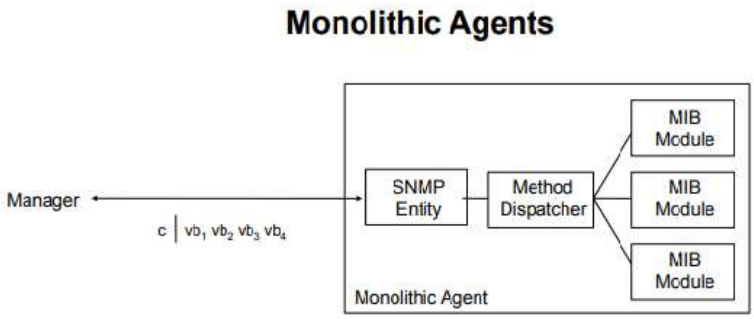
Versione di SNMP che **risolve i problemi relativi alla sicurezza** ancora presenti in SNMPv1 e SNMPv2. La versione 3 ha tentato di rimanere semplice, senza però riuscirci, ed il formato dei pacchetti è diverso rispetto alle versioni 1 e 2, poiché sono presenti anche campi relativi alla sicurezza.

SNMPv3 realizza un **controllo sull'autenticità e l'integrità del messaggio**, utilizzando il **protocollo CRC a livello Ethernet e la crittografia tramite un cifrario hash con chiave simmetrica**, nota a mittente e destinatario. Attraverso questo controllo siamo quindi in grado di sapere chi ha mandato il messaggio e controllarne l'autenticità.

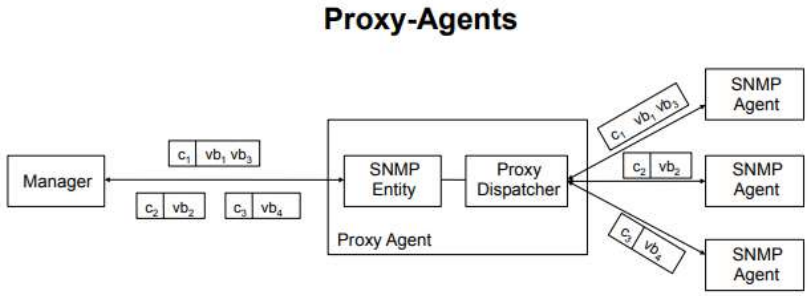
Si è costruito anche un **sistema di protezione per quanto riguarda la ripetizione di messaggi**, cosa molto comune in numerosi attacchi malevoli. Per fare ciò si utilizza un **meccanismo basato su un timestamp** inserito all'interno del messaggio da inviare: il destinatario, una volta ricevuto, controllerà se il timestamp ha una validità temporale corretta.

I **tipi di agent** in SNMPv3 sono:

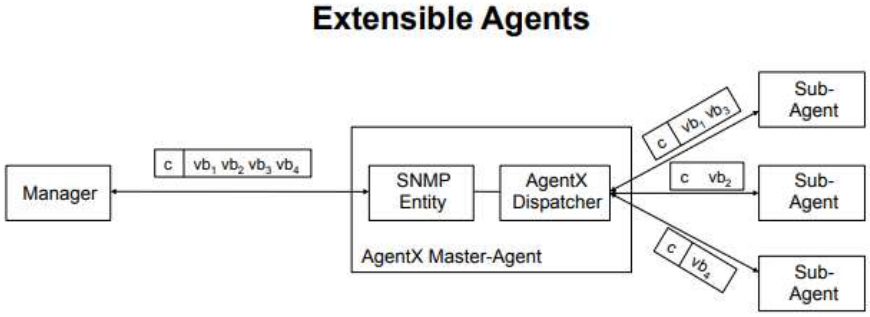
- **Agent monolitico:** implementato da un **singolo processo** contenente SNMP ed i MIB. È necessario un dispatcher intermedio per far arrivare le richieste ad un MIB specifico



- **Proxy agent:** consente di accedere agli agent non direttamente accessibili, ad esempio dietro un firewall. Il **proxy agent riceve le richieste** dal manager **e le inoltra ad uno specifico agent**. Non posso fare un'unica operazione passando come argomenti OID di agent diversi

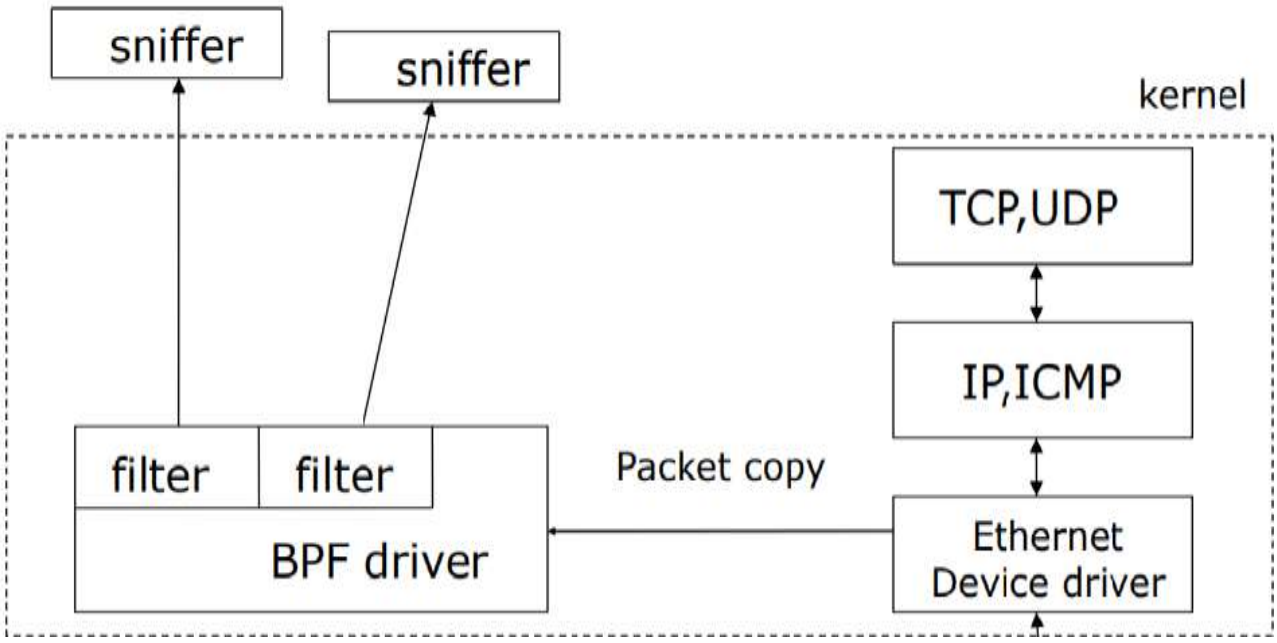


- **Agent X:** supporta il **concetto di master agent e sub-agents**. Il master agent non ha informazioni al suo interno, è un qualcosa sempre attivo a cui i sub-agents si registrano o cancellano. **Sta ai sub-agents fornire una lista dei propri OID** in modo tale che, quando il master agent riceve una richiesta, esegue un controllo sul messaggio e lo inoltra al sub-agent corretto. In questo caso si potrà anche inserire più OID relativi ad agents diversi all'interno della stessa richiesta



NETWORK MONITORING

Un'applicazione detta **sniffer**, per poter catturare il traffico dati della rete, **deve avere le giuste capabilities o i diritti di super utente root per poter vedere il traffico generato da tutti i processi in esecuzione sulla macchina**. Una **capability** è implementata come una **struttura dati privilegiata che consiste in una sezione che specifica i diritti di accesso e una sezione che identifica univocamente l'oggetto**. Le capabilities hanno come obiettivo la sicurezza e, in un sistema basato su di esse, si potrà utilizzare un particolare oggetto riferito in conformità con i permessi contenuti nella capability stessa. Infatti, **assegnare i diritti di super utente ad un'app potrebbe essere troppo rischioso** poiché le darebbe **troppa libertà**. Le **capabilities** quindi **servono per conferire privilegi ad un'app ma solo su una determinata area**, si possono vedere come **permessi di super utente limitati** ad un particolare ambito. Uno sniffer **opera su una shallow copy di un pacchetto** mantenendo dei contatori, quindi tutte le copie di uno stesso pacchetto puntano alla stessa area di memoria. Le **copie** dei pacchetti vengono **inviate al driver BPF**, il quale fornisce un'interfaccia a livello 2 per la ricezione dei pacchetti e supporta la modalità promiscua. Gli sniffer potrebbero non voler ricevere tutti i pacchetti, quindi il driver BPF **gestisce i pacchetti di interesse per gli sniffer**, i quali specificano i pacchetti da catturare tramite dei filtri, che vengono posti il più vicino possibile alla sorgente di traffico.



Solitamente le metriche di rete non sono standardizzate e ciò rende difficile effettuare confronti. **RFC1242** definisce lo **standard per le misurazioni** ed è esteso da altri RFC. Sono definiti traffic burst (traffico non costante, alterna picchi alti con periodi regolari e bassi di traffico), network load, traffico broadcast e altre metriche per il multicast come forwarding, throughput e overhead dovuto a join/leave di un gruppo.

Una delle prime e principali metriche da analizzare riguarda la **disponibilità di un servizio o availability** , ossia la **percentuale di tempo in cui il servizio è disponibile**: si calcola come un rapporto tra il tempo medio tra due fallimenti (MTBF) e la somma tra tale tempo ed il tempo di un recupero da un fallimento (MTTR). La formula sarà **100*MTBF / (MTBF+MTTR)**.

Altra metrica importante è il **response time**, che mi dice il **tempo di risposta tra client e server**. Si usa una sonda per misurarla e consiste nella **somma tra delay client e delay server**. Il **delay server** è **calcolato a partire dalla ricezione dell'ack** sulla sonda **fino alla ricezione del syn + ack**. Il **delay client è calcolato dal syn + ack all'ack inviato dal client**.

DEEP PACKET INSPECTION (DPI)

L'obiettivo dell'analisi del traffico di rete consiste nel comprendere che tipo di traffico sta transitando sulla rete in un dato momento. Prima dell'avvento di DPI le principali tecniche di analisi del traffico di rete sono state:

- **TCP/UDP port classification:** nei primi anni di Internet i **protocolli di rete** erano **identificati dalle porte utilizzate** (in Linux nel file etc/services). Questa tecnica **consente d**i **individuare solo quei protocolli che operano su porte note**, ma in caso di attacchi malevoli è un fenomeno facile da aggirare
- **QoS-based classification:** simile alla classificazione basata sulle porte, in questo caso **ci si basa su alcuni tag QoS** presenti nei pacchetti in transito. Anche questa tecnica è però facile da alterare e aggirare
- **Statistical traffic classification:** classificazione di pacchetti IP e flussi basandosi su regole stabilite a priori o utilizzando **tecniche di machine learning**. Il problema dovuto all'utilizzo di algoritmi di ML è che essi **richiedono dei training set accurati ed un'alta capacità computazionale**. In più, offrono un'alta precisione sui dati di training ma **si potrà avere degli errori nella classificazione di nuove istanze**

Con **DPI** si va ad **esaminare il payload di ciascun pacchetto**, creando un possibile problema riguardo alla sicurezza e al contenuto del pacchetto, che viene esposto. Dato che **per una classificazione accurata diventa importante** andare a vedere il contenuto di ogni pacchetto, questo aspetto passa in secondo piano. La scelta di adottare DPI come tecnica per l'analisi del traffico di rete è dovuta al fatto che analizzare gli header non fornisce molte informazioni utili alla classificazione, in più i gestori di una rete vorrebbero classificare il traffico indipendentemente dalle porte in uso.

nDPI è un toolkit opensource che usa DPI per analizzare e classificare il traffico di rete e supporta circa 250 protocolli diversi. nDPI **identifica un protocollo come <major>.<minor>** dove **major** è un **protocollo di rete** e **minor** un **protocollo applicativo**. Solitamente non si basa su un singolo protocollo ma su una famiglia di protocolli, suddivisi per categoria (es. http/tls, social, cloud … sicuri, insicuri …). Quando uno sniffer ha catturato il traffico, lo passa a nDPI che tenta di classificarlo. Questo processo viene realizzato tramite dei **dissectors**, ossia **file in C** che decodificano il traffico e tentano di classificarlo. Sono suddivisi in due macro-categorie (TCP e UDP) e **per ciascun protocollo specifico ce n'è uno**. Quindi **in base al tipo di traffico**, utilizzerò i

dissectors di una specifica categoria partendo da quello che ha più probabilità di fare match. Questo meccanismo è reso efficiente dal **caching**, ossia un **principio secondo il quale posso applicare un dissector già usato al traffico da classificare se questo è simile a del traffico già analizzato e catalogato**. Potrebbe succedere di non avere un match poiché il protocollo non è supportato oppure perché nDPI si è perso la fase iniziale di una data sessione. Quindi se un dissector non fa match dopo un certo numero di pacchetti visti di una data sessione, non si continua ad ispezionare i pacchetti successivi perché si potrebbe ricadere in falsi positivi. Infatti, **nDPI classifica il traffico basandosi sull'analisi dei primi 8 pacchetti di una sessione**: quindi perdersi la sessione iniziale, cioè questi 8 pacchetti, non permette di classificare quella tipologia di traffico.

Il **traffico catturato** viene **mantenuto in una flow table presente nell'applicazione o nel sistema**. Ogni **flusso** è bidirezionale e **denotato da 5/6 componenti**: IPsrc, Psrc, IPdst, Pdst e tag vlan. Quando si riceve il **traffico classificato** da nDPI, **l'applicazione andrà a controllare all'interno della flow table se esiste già un flusso per quel particolare tipo di traffico**. Questa tabella è gestita come una tabella hash e ad intervalli regolari viene ripulita dai flussi terminati o inattivi. Le **performance di nDPI** sono **influenzate dal n° di pacchetti da analizzare e dal n° di flussi concorrenti**. Questi ultimi sono un fenomeno di delicata gestione, poiché rappresentano quei tipi di traffico che si differenziano per un solo componente della quintupla (es. DNS), creano collisioni nella flow table e aumentano il carico di lavoro di nDPI.

MISURE DELLA RETE

- **Throughput:** quantità di dati presenti su un collegamento in un dato intervallo di tempo. È una misura applicativa e ci da la stima della banda disponibile in un dato istante
- **Goodput:** simile al throughput, ma considera solamente i bytes del payload di un pacchetto
- **Latenza:** tempo necessario ad un pacchetto per andare da sorgente a destinazione, è una misura monodirezionale
- **Round Trip Time (RTT):** tempo necessario ad un pacchetto per andare da sorgente a destinazione e viceversa (latenza da A a B + latenza da B a A)
- **Jitter:** varianza del ritardo tra sorgente e destinazione, da un'idea di come varia la latenza nel tempo. Il suo valore ideale è 0, mentre se è dell'ordine della latenza avremo una rete che procede a scatti
- **Banda:** quantità di dati trasferiti su un link in un'unità di tempo, in genere bps. Si differenzia in 2 tipi:
  - **Burst committed (Bc):** massimo numero di bit che l'operatore garantisce di trasferire
  - **Burst exceed (Be):** banda che l'operatore fornisce se avanza

  Si calcola poi il MDR (Maximum Data Rate) come (Bc + Be) / time

- **Misure per link:** misure fatte sul singolo collegamento, non forniscono statistiche globali
- **Misure end to end:** misure fatte da sorgente a destinazione, l'osservazione sarà meno dettagliata

- **Monitoraggio passivo:** si osserva ciò che passa sulla rete, ci dovrà essere qualcosa in transito altrimenti non vedo niente
- **Monitoraggio attivo:** tecnica più intrusiva, si iniettano dei pacchetti nella rete e si osserva cosa succede (es. ping). Essendo una tecnica invasiva bisogna fare attenzione al traffico generato
- **Misurazione inline:** la rete è la stessa utilizzata sia dagli applicativi che dallo sniffer che fa il monitoraggio
- **Misurazione offline**: chi fa il monitoraggio utilizza una rete diversa rispetto a quella su cui fare l'analisi

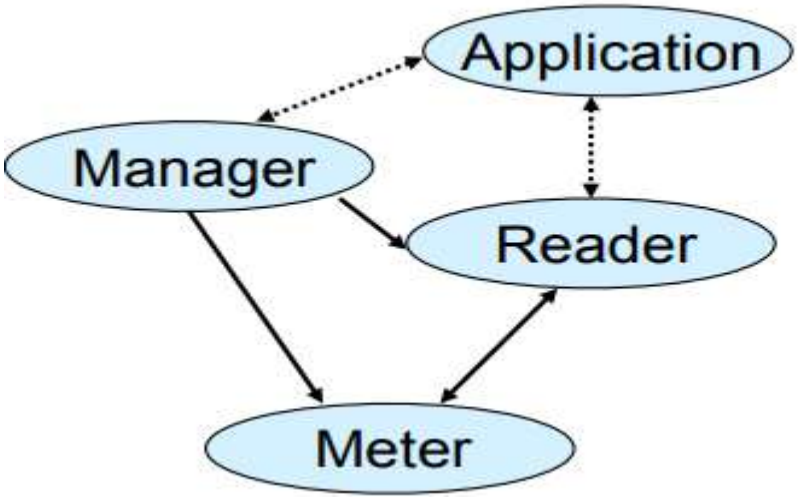
MONITORAGGIO REMOTO CON SNMP

L'idea di **monitorare reti remote con SNMP** nacque con il **MIB RMON**, di cui esistono due versioni che si occupano del livello 2 e 3. RMON consente di monitorare il traffico su una rete remota, collezionare dati ed inviarli periodicamente ad un ente centrale di gestione. L'idea è di lasciare le decisioni da prendere all'agent, andando a ridurre il carico di lavoro del manager.

All'interno di RMON **ci sono diversi gruppi che danno informazioni e statistiche sui device**: **statistics** (info sulle interfacce), **history** (breve andamento del device), **alarm** (soglie per controllo eventi anomali), **host** (statistiche per ogni host della rete), **host top N** (statistiche per i primi N host della rete), **matrix** (matrice di flussi), etc… Una metrica importante calcolata da RMON è la **network utilisation**, calcolabile **per ogni porta di un device** e **utile per tracciare un andamento nel tempo su una specifica porta**. Si calcola come **(#pacchetti*160*100 + #ottetti*8) / (velocità*tempo).**

Negli anni RMON ha stimolato il **progresso nell'analisi del traffico di reti remote**, con modalità via via più efficienti. Un esempio è **RTFM** (Real Time Flow Measurements), la cui **architettura** è composta da:

- **Meter:** sta dove passa il traffico e lo misura, è una specie di sonda
- **Reader:** estrapola i dati dal meter e li esamina, dopodichè li invia all'applicazione finale
- **Manager:** gestisce il reader e setta le politiche del meter
- **Applicazione**



MEDIA, VARIANZA E DEVIAZIONE STANDARD

In generale, quando si fa una misurazione, diventa complicato stabilire cosa è grande e cosa è piccolo senza un metro di paragone.

La **varianza** è la **media delle differenze dalla media al quadrato** e ci dice **come varia una grandezza nel tempo**, mentre la **deviazione standard** è la **radice quadrata della varianza** e ci dice **di quanto una misura si discosta dalla media**. Quindi **la deviazione standard ci fornisce un mezzo attraverso il quale possiamo stabilire se una data rilevazione è ok**, **troppo bassa** (< media - dev.) **o troppo alta** (> media + dev.).

Il **percentile** è un **valore al di sotto del quale ricade una certa percentuale di osservazioni**: per il **monitoraggio della rete** si utilizza il **95° percentile**, ossia il valore al di sotto del quale ricade il 95% delle osservazioni di un dato fenomeno. Nell'analisi della rete il 95° percentile è utile perché ci permette di fare considerazioni sull'utilizzo della banda e sulla sua capacità.

SERIE TEMPORALI E PREDIZIONE

Una **serie** è una **sequenza ordinata di valori**, dove l'**ordine** è **dato dagli indici** che denotano gli elementi: quindi una **serie temporale** è una **sequenza di valori ordinati temporalmente**. Un'**osservazione** è un **valore numerico rilevato in un dato istante** riferito ad un fatto già accaduto, mentre la **predizione** è la **stima attesa di un valore che mi aspetto di vedere** in un istante futuro. Si definisce quindi **SSE** (Sum of Square Errors) come la somma delle differenze tra il valore predetto e quello osservato al quadrato. Il **valore ideale** di SSE sarebbe tendente a 0**,** significa che la **predizione fatta** è **corretta**.

Sia **y^x** la **predizione del valore y al tempo x**, esistono diverse modalità per calcolarla:

- **Media semplice:** il prossimo valore si ottiene dalla media dei precedenti
- **Media mobile:** il calcolo del prossimo valore si basa sugli ultimi n valori rilevanti
- **Media mobile pesata:** si utilizzano dei coefficienti per dare più o meno peso a valori recenti o più vecchi
- **Single exponential smoothing: $y\hat{}x = \alpha * yx + (1-\alpha) * y\hat{}x-1$**.
  È una formula ricorsiva che restituisce un singolo valore, dove **α** è un **coefficiente tra 0 e 1 detto fattore di decadimento**. Quindi α **stabilisce quanto i dati passati mi influenzano la predizione**, se è prossimo a 1 considero di più il recente passato, se tende a 0 utilizzo un approccio più conservativo.
- **Double exponential smoothing:**
  $$y\hat{}x = \alpha * yx + (1-\alpha) * y\hat{}x-1$$
  $$bx = \beta * (y\hat{}x - y\hat{}x-1) + (1- \beta) * bx-1$$
  $$y\hat{}x+1 = y\hat{}x + bx$$
  Si utilizza una formula ricorsiva in cui **si tiene conto del trend b e di un altro coefficiente β**, detto **fattore di trend**, che ha un ruolo simile a α e **tiene conto dell'andamento al passo precedente**. In questo caso **si restituiscono due valori predetti**, cioè **y^x e y^x+1**

- **Triple exponential smoothing:** si basa sull'**algoritmo Holt-Winters** e definisce alcuni nuovi concetti. Una **stagione** è una **serie ripetuta ad intervalli di tempo regolari** con una durata. La formula è sempre ricorsiva e considera un altro coefficiente **γ**, detto **fattore di stagionalità**. La **formula restituisce una serie di valori predetti con un andamento simile al passato** ma possibilmente soggetti ad errori. I **coefficienti α, β e γ** sono i **valori adattativi** dell'algoritmo HW, tutti **compresi tra 0 e 1**: valori alti significano che l'algoritmo si adatta velocemente, al contrario valori prossimi a 0 rendono l'algoritmo conservativo e legato al passato.

NETFLOW

Un **flusso** è un **insieme di pacchetti che presentano caratteristiche in comune** (es. quintupla) e può essere monodirezionale o bidirezionale. I flussi **contengono informazioni sul traffico della rete** ma non sul device usato per il monitoraggio, detto sonda o probe. Le **informazioni repereibili da un flusso** sono **n° di pacchetti** che lo compongono, **tempo di inizio e fine flusso**, **interfacce di entrata e uscita del router**, **indirizzi IP** sorgente e destinazione, **porte**, **informazioni di routing**, etc… Un **flusso** viene **creato** quando non esistono pacchetti simili e viene **mantenuto in una tabella detta NetFlow cache**. Due flussi monodirezionali opposti sono un unico flusso bidirezionale, su cui ho alcune informazioni aggiuntive. Mantenere flussi separati aumenta la granularità delle informazioni ma anche il carico di lavoro della CPU e del device per il monitoraggio.

Il **probe** presente sul router **non ha capacità di memorizzazione**, quindi un flusso quando termina o resta inattivo viene esportato. I **flussi** sono **esportati verso un collector**, che ha il compito di **assemblare ed interpretare i flussi ricevuti** al fine di **stilare un report** utile al gestore della rete. Generalmente, **in reti di dimensioni elevate** con molti router e probe, quello che si fa è **mantenere i flussi esportati in un collector locale** per evitare che le informazioni viaggino in rete o siano mantenute tutte in un collector centrale. Inoltre, **per evitare la duplicazione del traffico**, nelle reti solitamente **si posizionano i probe sui router di confine e su di essi si monitora solo il traffico in entrata e fino al livello 3.**

Negli anni sono state realizzate diverse versioni di NetFlow, ognuna con il proprio formato di pacchetto. Ogni pacchetto inoltrato dal router viene analizzato secondo i criteri stabiliti e si va a controllare se nella cache è già presente un flusso per quelle particolari caratteristiche. In base ai criteri scelti può cambiare il n° di flussi che si vanno a generare. Un **flusso termina** quando si trova il **flag di fine sessione**, **dura troppo**, **resta inattivo troppo a lungo** o si è **raggiunto la capienza massima della cache**. Il software all'interno del router andrà ad esaminare la cache per trovare i flussi da esportare verso il collector. Un flusso lungo potrà essere suddiviso in tanti sotto-flussi che verranno riassemblati dal collector. Il motivo per cui i flussi hanno una durata massima è dovuto al fatto che **non si può trasferire il flusso alla fine della sessione, ma si rende necessario fornire aggiornamenti continui**. Infatti, il collector necessita di dati sempre aggiornati da mostrare alle applicazioni a lui collegate. **L'esportazione dei dati dipende dalla configurazione del probe** e si basa su un **protocollo connectionless** per essere veloce, leggera ed indipendente da altre entità coinvolte nella comunicazione.

In genere, **un probe al suo interno ha un agent SNMP** e vale che il **traffico SNMP è maggiore o uguale al traffico NetFlow**. Questo perché **SNMP tratta informazioni di livello 2**, quindi **considera anche pacchetti multicast e broadcast** che non escono dalla rete **ed il payload di livello 2**. In **NetFlow** si considera il **traffico a livello 3** e si esamina solo il traffico **che transita su un router**, ossia che viene inoltrato **da un'interfaccia di input ad una di output**.

NETFLOW v5 PACKET FORMAT

Nei pacchetti di ogni versione ci sono campi comuni e campi specifici per ciascuna versione. In un **pacchetto di NetFlow** v5 si ha un **header ed un insieme di flussi esportabili, detti flow records** (nella v5 max. 30). L'**header di un pacchetto NetFlow** contiene

- versione corrente di NetFlow
- n° di flussi del PDU
- sysuptime del router
- ora in cui i flussi sono stati esportati
- numero di sequenza utile al collector per capire se ha perso flussi precedenti
- engine_type e engine_id

Il **contenuto di un flow record** è invece formato da:

- indirizzi IP sorgente e destinazione
- next hop
- id delle interfacce di input e output del router
- n° di pacchetti e bytes inviati
- tempo del primo e dell'ultimo pacchetto del flusso
- porte sorgente e destinazione (se è un protocollo senza porte si mette 0)
- or dei flag TCP
- protocollo di livello di trasporto
- AS sorgente e AS destinazione
- netmask dell'IP sorgente e destinazione

NETFLOW v9

I **formati fissi dei pacchetti** furono definiti in **NetFlow dalla v1 alla v8**, in ogni versione sono state introdotte delle piccole modifiche, ma si è reso necessario ridefinire un formato più libero e flessibile. Inoltre, si volevano introdurre modalità di supporto per il livello 2, vlan, IPv6, etc…

Con **NetFlow v9** si è deciso di introdurre il concetto in cui il **formato del pacchetto** viene **deciso dal probe in base ai dati che deve esportare** in quel momento. Il **formato** è **detto template** e **specifica la codifica dei dati e la loro struttura al collector**, il quale non potrà più basare le proprie operazioni su una struttura fissa standard come nelle precedenti versioni. Il nuovo

pacchetto in NetFlow v9 è quindi **composto da un header, un flow template e i flow records**. **Se il collector** per qualche motivo **non riceve il flow template non sarà in grado di decodificare i dati**, quindi il **template** viene **inviato ogni n secondi oppure ogni n flussi**. Altri due campi sono option template e option record, che forniscono informazioni sulla configurazione del probe.

In **situazioni in cui il traffico da analizzare è elevato** possiamo andare ad operare un fenomeno detto **sampling**. Esso è utile perché **permette di ridurre il carico di lavoro della CPU e l'utilizzo della memoria**, a patto di perdere alcune informazioni. Esistono **2 tipologie** di sampling:

- **Packet sampling rate:** indica quanti pacchetti considero sul totale (es. 1:100). **Serve a ricevere meno pacchetti da analizzare**, potrei però ottenere dati non corretti poiché **i flussi che vado a creare avranno i contatori dei bytes sballati**. Si utilizza in casi limite dove ci basta avere un'approssimazione del traffico suddiviso in flussi e **si usa per alleggerire il carico di lavoro del probe**
- **Flow sampling rate: limita il n° di flussi da esportare**, i pacchetti vengono ricevuti e analizzati tutti. In seguito, **si esportano meno flussi di quelli che abbiamo per alleggerire il carico di lavoro del collector.**

STRUTTURE DATI EFFICIENTI

- **Bitmap:** array di bit di lunghezza arbitraria in cui aggiunta e rimozione costano O(1)
- **Compressed bitmap:** bitmap compresse **utilizzate per rappresentare insiemi sparsi in cui si hanno solo pochi bit settati a 1**. Esistono alcuni algoritmi di compressione come WAH, EWAH e COMPAX, i quali però costruiscono la bitmap compressa in modo incrementale. Queste strutture dati **rendono efficienti operazioni come AND, OR e NOT senza bisogno di decomprimere la bitmap**
- **Bloom filters: struttura dati probabilistica utile a verificare se un certo elemento fa parte di un insieme**, con costo e velocità efficienti. Sono vettori di bit di lunghezza m che **sfruttano due funzioni hash indipendenti**. Per aggiungere un elemento x all'insieme si settano a 1 i bit di h1(x) e h2(x), mentre **per vedere se un dato elemento x appartiene all'insieme si controllano i bit in corrispondenza di h1(x) e h2(x)**. **Se entrambi sono a 1 allora x probabilmente appartiene all'insieme**, **la probabilità è dovuta al fatto che si possono verificare delle collisioni a causa di h1 e h2**. Esiste una variante chiamata counting bloom filter che al posto del bit utilizza un contatore, in modo da realizzare anche la rimozione dall'insieme
- **Tries:** struttura dati ad albero non basato su confronti in cui **ogni nodo ha una lettera e in caso di più opzioni si ha una lista**. I nodi non conservano una copia della loro chiave, la quale dipende dalla posizione nell'albero. I nodi possono essere aggiunti, rimossi e anche cercati. Si possono cercare stringhe che iniziano con un dato prefisso, dato che tutti i discendenti di un nodo condividono con esso lo stesso prefisso.

- **Radix tree: simili ai trie, alcuni nodi hanno un insieme di lettere e possono collassare insieme se hanno stessa radice**. Uno tra i più importanti radix tree è il **Patricia tree**, in cui al posto delle lettere si usano i numeri. Sono **utilizzati per rappresentare reti e sottoreti** e per verificare l'appartenenza di indirizzi IP ad esse.

IPFIX

Il **problema di NetFlow** è sempre stato quello di essere un **prodotto della Cisco**, quindi di avere un **formato proprietario** e un RFC. L'**obiettivo** è diventato quindi quello di **definire un formato standard e generico simile a NetFlow**: questo formato è stato definito da IETF e si chiama **IPFIX**.

Tramite IPFIX **si può definire nuovi flow fields utilizzando un formato simile ad un OID**, in modo da avere la sicurezza che il campo appena definito sia univoco. Il **protocollo utilizzato per esportare i flussi è SCTP** (Stream Control Transport Protocol), un ibrido tra TCP e UDP.

FLOW AGGREGATION E FLOW FILTERING

In generale i **flussi raw** sono **utili ma limitanti**, nel senso che non rispondono a domande del tipo quanto traffico è web/mail, quanto traffico va verso il provider x, etc … Per rispondere a queste domande ci interessa effettuare delle **aggregazioni**, che possono essere fatte **sia dal collector che dal probe.**

- **Probe aggregation: si aggregano i flussi secondo criteri stabiliti dall'utente**, andando a migliorare l'utilizzo della memoria per quanto riguarda la Netflow cache. Di contro **il collector non potrà risalire alle informazioni dei singoli flussi** iniziali
- **Collector aggregation: l'aggregazione viene fatta dal collector**, che dovrà svolgere più lavoro ma potrà prima salvarsi i singoli flussi per risalire alle informazioni in essi contenute

Prima di effettuare l'aggregazione dei flussi, si potrà effettuare un **filtraggio** secondo criteri stabiliti dall'utente. Il filtraggio è un **processo diverso dall'aggregazione, poiché esso si applica prima di aggregare i flussi.**

SFLOW

NetFlow per sua natura è **nato per i router e per traffico** che generalmente viene routato, ossia **intra-LAN**. Quindi il suo grosso limite è appunto quello di essere **nato per analizzare il traffico in reti WAN o geografiche**, non si vede il traffico LAN. Si potrebbe quindi mettere una sonda su ciascun device per analizzare il traffico LAN. I **principali problemi** potrebbero essere **l'ammontare di traffico da analizzare**, il **costo dei dispositivi di rete** utilizzati per il monitoraggio, l'**aumentare della velocità delle reti**, **scalabilità**, etc…

Quindi entra in gioco **sFlow**, il quale non parte dal principio di NetFlow per cui va visto tutto il traffico che transita. Si può applicare **sia su reti WAN che su reti LAN** ed i **principi di sFlow** sono i seguenti:

- **Non pretende di essere veloce quanto la rete monitorata**, poiché anche se riuscisse a rilevare tutto il traffico, avrebbe comunque problemi nel gestire i flussi generati
- Effettua il **sampling sui pacchetti**, sapendo che più pacchetti vede più i suoi report saranno precisi. Il sampling **si effettua su tutte le porte del device ed in modo da avere sempre la stessa proporzione ma in modo casuale**, per non rischiare di avere un traffico periodico e di analizzare solo quello
- Se la **velocità della rete** è comunque **alta**, applica un **sampling maggiore**
- La **perdita dei pacchetti** e del traffico deve essere **controllata**, nel senso che va saputo il motivo di quella perdita

L'**architettura** su cui si basa sFlow è composta da un **probe che cattura il traffico**, da un **collector che riceve i pacchetti esportati** ed **un'applicazione che riceverà i report**. Il **probe sFlow** opera secondo principi diversi da quelli del probe NetFlow, poiché **prende un pacchetto da ciascuna porta dello switch su cui si trova in base al sampling rate che si sta applicando**. Una volta presi i pacchetti, **li incapsula e li invia al collector, quindi il probe non analizza il pacchetto ma si limita ad inviarlo al collector**: è questo modo di fare che influisce sulle prestazioni e consente di andare più veloce.

Inoltre**, sFlow consente di risolvere un problema di SNMP**. Esso infatti fornisce solo informazioni quantitative, quindi **gli sviluppatori di sFlow hanno deciso di inserire all'interno dei pacchetti sFlow anche alcune statistiche e contatori del MIB-II riguardanti le interfacce di rete** del collector, detti **counter samples**. In questo modo non c'è più bisogno di andare a fare polling sullo switch. sFlow è descritto nell'RFC3176 e ha varie versioni, la più usata è la v5. Inoltre, essendo in esecuzione sullo switch, nel pacchetto sFlow si può trovare informazioni ulteriori sullo switch stesso o in più rispetto alle statistiche di SNMP.

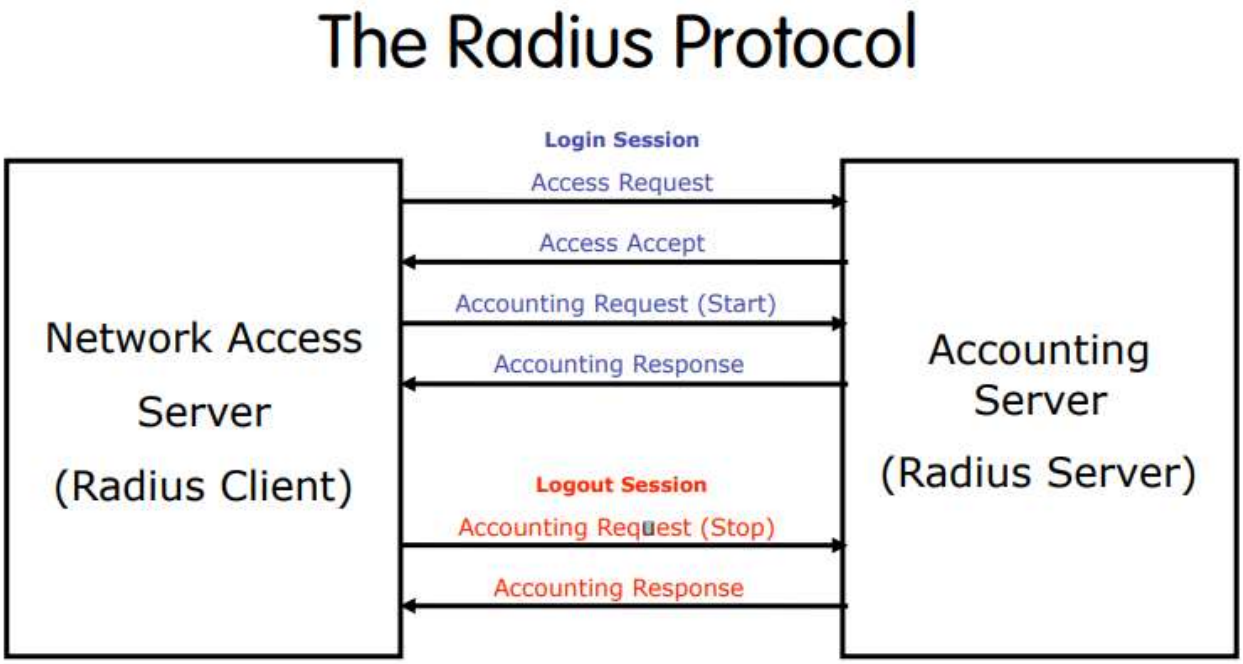
Quindi **in NetFlow un flusso è generalmente un insieme di pacchetti con stessa quintupla**, mentre **in sFlow un flusso è solamente un pacchetto, che viene preso dal probe ed inviato al collector senza essere analizzato**. Un pacchetto sFlow è composto da:

- Versione di sFlow in uso
- Indirizzo IP dell'agent che ha catturato ed esportato il pacchetto (serve perché il traffico potrebbe passare attraverso un NAT, etc…)

- Numero di sequenza per sapere se ho perso qualche pacchetto
- Numero di samples
- Sysuptime dello switch
- Flow sample (pacchetto)

PROTOCOLLO RADIUS

Radius sta per **Remote Authentication Dial In User Service** ed è importante perché è il **protocollo più usato per l'autenticazione di un device su reti sia fisse che mobili** (Diameter, evoluzione del Radius). È un protocollo che non sta sui PC, smartphone, router o altro, ma è lato operatore.



Il **Radius Client** si trova **all'interno della centrale del provider**, non è lato utente finale, e **quando l'utente si vuole connettere alla rete manda una richiesta al Radius Server** che solitamente è un **software collocato nei datacenter del provider**. Il Radius Server ha accesso all'anagrafica degli utenti e quindi può accettare o rifiutare la richiesta del Radius Client. Dopo questa sessione di accesso, il Radius Client manda le richieste dell'utente alle quali seguiranno risposte da parte del Radius Server, poi alla fine della sessione si può disconnetterci in modo pulito o bruscamente.