Gestione di Reti

Federico Matteoni

A.A. 2019/20

Indice

Introduzione

Perché bisogna studiare la gestione? La situazione corrente comprende: un aumento delle risorse strategiche informative, le reti di computer che da strumento di supporto sono diventate elemento chiave delle organizzazioni, l'aumento esponenziale dei dispositivi interconnessi e aumento anche della complessità e delle funzionalità.
C'è quindi richiesta di servizi di rete permanenti e di qualità ottimale, oltre alla necessità di ridurre i costi per le infrastrutture di rete di un'azienda.

Necessità Gestione di reti eterogenee con l'aiuto dei computer.

Terminologia e concetti fondamentali

Managed Objects Il controllo, la coordinazione e il monitoraggio delle risorse avviene tramite la manipolazione dei cosiddetti **managed objects**: un MO è una visione astratta di una risorsa che presenta le proprietà dal punto di vista della gestione. Sono **rappresentazioni astratte di risorse reali**.
I confini di un MO specificano quali dettagli sono accessibili ai sistemi di monitoraggio e quali sono schermati (**black box**)

Management-System ↔ Managed Object ↔ Real Object

Caratteristiche

Attributi: descrivono lo stato/condizione dell'MO, possono cambiare quando cambia lo stato dell'oggetto reale e possono essere manipolati attraverso operazioni di management

Operazioni: consentono l'accesso all'MO. Operazioni tipiche sono get, set, create e delete, ma il numero e tipo delle operazioni influenzano perfomance e complessità dell'oggetto

Comportamento: determina la semantica e l'interazione con la risorsa reale. Normalmente definito in linguaggio naturale

Notifiche: quantità e tipologia dei messaggi, che possono essere generati da situazioni pre-definite da un MO quando avviene una specifica situazione

Management Information Base L'unione di tutti i MO contenuti in un sistema forma la MIB del sistema. La **Management Information Base** è la collezione di tutti i management object all'interno del sistema, con i loro attributi.
Una MIB deve essere conosciuta sia da chi la implementa che da chi la gestisce.

Modularità Gli MO di un sistema sono solitamente definiti in più MIB. Nelle MIB sono introdotti i moduli per consentire un design modulare: moduli diversi possono essere definiti da team diversi, le funzionalità di gestione possono essere estese e modificate...

Paradigma Gestore/Agente Agent

Implementa i MIB delle MO accedendo alle risorse reali

Riceve le richieste da un gestore, le processa e trasmette le risposte appropriate

Smista le notifica riguardanti cambiamenti di stato importanti nel MIB

Protegge gli MO da accessi non autorizzati usando regole di controllo degli accessi e autenticazione della comunicazione

Manager

Esercita il controllo delle funzioni

Avvia operazioni di gestione tramite opportune operazioni del protocollo per la manipolazione degli MO

Riceve messaggi dagli agenti e li inoltra alle applicazioni interessate per la gestione

Management Protocol Un protocollo di gestione implementa l'accesso a MO distanti attraverso la codifica di dati di gestione (management data)

0.1 lezione 2

Livello 2 consente di identificare un device sulla rete. In tutte le reti c'è la necessità di identificare la porta di rete. Ogni dispositivo ha almeno un'interfaccia di rete: loopback, che consente di far comunicare processi di rete sulla stessa macchina. `127.0.0.1` consente di parlare su stessa macchina senza trasmettere sul filo, fondamentalmente un cortocircuito.
`ifconfig` consente di vedere le interfacce di rete disponibili su unix.
Se si vuole gestire una rete è fondamentale la standardizzazione.

Output `ifconfig`. Parte degli indirizzi, no indirizzo hw su loopback perché il traffico non esce mai (loopback sulla pila OSI è nel livello 3 Network, il MAC address è sul livello 2 Data Link, che non viene toccato da loopback). Indirizzo MAC 6 byte divisi in blocchi dai due punti. I primi 3 identificano il costruttore della scheda di rete. I successivi tre identificano la scheda di rete per il costruttore, che lo setta univocamente. Ciò garantisce univocità. Per primo blocco di tre ho 16M di dispositivi possibili. I MAC address quindi **non sono univoci**, lo sono *probabilmente*. L'univocità è fondamentale sulla stessa rete. Quindi indirizzo hw identifica univocamente device sulla rete locale. divisi in due blocchi, il primo identifica costruttore della scheda di rete.

Qualsiasi dispositivo ha indirizzo hw diverso per ciascuna porta.

0.2 Ethernet

Ethernet è un cavo seriale, trasmissione e ricezione. Mezzo seriale. Un filo.
Quando si mandano dati non posso tutti insieme ma man mano. Non c'è collisione perché ricezione e trasmissione sono su due fili separati.
Pacchetti inviati nel tempo sul filo. Vengono distinti tra loro dal `preamble`. Pacchetti inviati in una direzione: preambolo, destinazione, sorgente, tipo dei dati, dati effettivi, padding (per rendere pacchetto di 64 se pacchetto è troppo corto), CRC.
Quindi per spedire pacchetto necessito di indirizzi (chi voglio e chi sono) e cosa mandare. chi sono lo so, è scritto nella scheda. Voglio conoscere indirizzo di chi voglio.
Alla connessione del cavo, se DHCP manda fuori pacchetto per richiesta quindi switch lo impara, se IP statico manda pacchetto ARP quindi switch lo impara.
MAC address randomizzato per privacy, spesso e volentieri sui dispositivi mobili.
Possibile più di un utente sulla stessa rete con soliti indirizzi, apparati avanzati se ne accorgono.

0.3 lezione 3

un pacchetto è interamente creato dal computer, quindi "non ci si può fidare"
Bisogna andare a livello fisico e autenticare, un po' come chiedere la carta d'identità. Metter in atto meccanismi che impediscano di inibire riconoscimento della sorgente.
802.1x permette di entrare in rete. Se configurato, il device prima di entrare in rete espone delle credenziali (utente, password, protocollo autenticazione...)
Da quel momento in poi **allegato** al pacchetto c'è il mio nome, ma le informazioni di autenticazione non fanno parte del pacchetto: pacchetti creati quando non c'era preoccupazione e interesse in fattori di sicurezza delle trasmissioni.
L'informazione non è parte del pacchetto ma lo riconosce in qualche altro modo il device e **rimane nel device** (Access Point). Ciò non serve per autenticazione fisica sul cavo: so che sei tu su questo cavo. Ma è necessaria per autenticazione su mezzi condivisi (wifi).
Su router MAC cambia ad ogni hop (ethernet comunicazione punto-punto), IP cambia solo se c'è NAT. Le parti da lv 3 in su non cambiano (a meno di frammentazioni...)
Robustezza delle reti si fa tramite la ridondanza. Tipico mettere più strade per spedire il traffico: load balancing.
Vale sia per corrente elettrica che per traffico di rete.

0.3.1 Aree funzionali

FCAPS per gestire **qualsiasi sistema**, da giochi a sistemi di rete. Non sono mutualmente indipendenti.

Fault Management: error detection, isolation and repair
Se qualcuno rileva malfunzionamento (riempito disco, ram, sovraccarico CPU...) lo deve notificare

Configuration Management: devo sapere com'è configurato il sistema. Leggere la configurazione è importante, così che le app si possano basare sulle API comuni e funzionare correttamente. Fondamentale capire la configurazione perché permette di definire l'amministrazione, servizi..., possibile riconoscere anche le adiacenze e "questo filo qui va su questa porta qua". che impatto ho se stacco questo cavo, o si rovina? Informazioni sufficenti per amministrare la rete

Account Management: rilevare il consumo di risorse

Performance Management: efficenze e statistiche, performance di sistema sia lato utente sia lato fornitore. Per l'utente è riuscire ad usare la rete, per l'operatore è il giusto compromesso tra investimento sul mezzo e contentezza utente.

Security Management: assicurarsi che ciò che uno fa è effettivamente possibile farlo, autoproteggendosi perché con le reti odierne posso intasare rete (volente o no) e quindi intasare internet, provocando danni

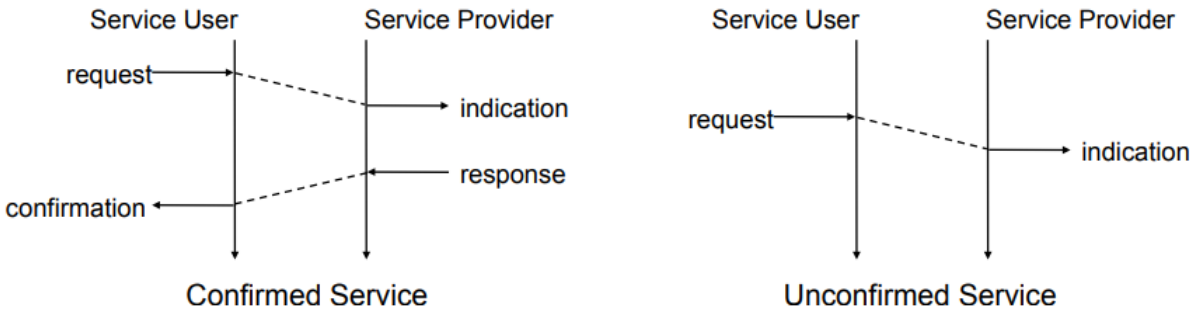
0.3.2 Interagire con management object

Primitive: get, set, create, delete
Quando faccio richiesta ad un protocollo mi aspetto una risposta: richiesta – risposta
Contenuto richieste varia durante il transito aggiungendo determinate informazioni. Es: SMS durante il transito aggiunge numero mittente per poter comunicare a destinatario chi inviava.

0.3.3 Servizi

Confermati Faccio richiesta → mi aspetto risposta.
Es: Telegram/Whatsapp

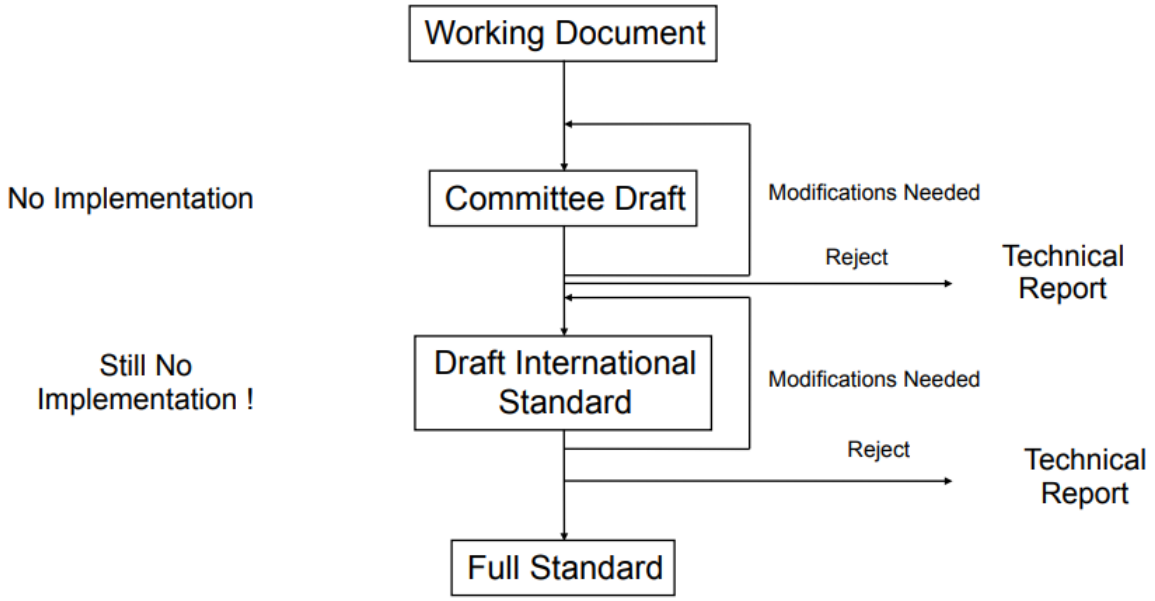
Non confermati Faccio richiesta e fine.
Es: SMS



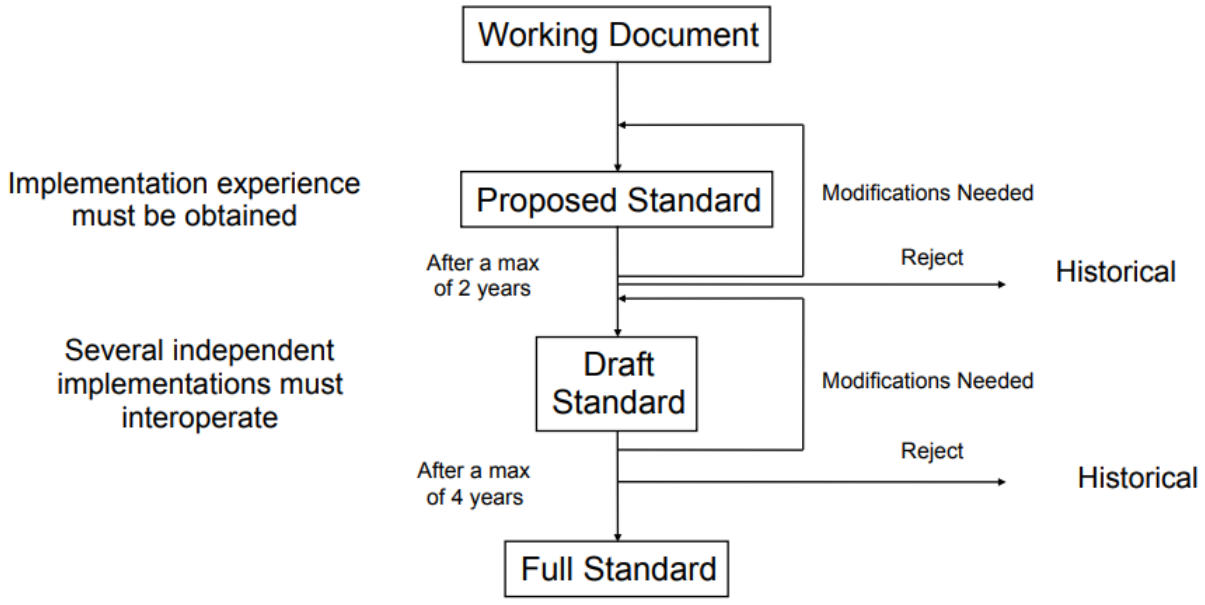
0.3.4 Standardizzazione

La grande differenza tra ISO/OSI e Internet è il processo di standardizzazione.

ISO Nella standardizzazione ISO tutte le varie aziende si accordano su come fare la rete: si creano gruppi di lavoro che si riuniscono (es. ICANN) e producono un documento di lavoro, poi vari comitati lo discutono (fino a qui **ad alto livello**).
Se si mettono d'accordo, pubblicano un **Draft International Standard** (**senza implementazione**). Dopodiché, se non viene accettato, creano technical report oppure un **full standard**.
La parte importante è l'assenza di implementazione fino ad avere la creazione dello standard.



Internet Nella standardizzazione internet è tutto seguito dal comitato IETF, che pubblica aree d'interesse in cui ritiene ci sia possibilità di sviluppo. La sottomissione di un'idea ad un'area d'interesse è libera, oppure si può mandare una mail per idee completamente nuove. Dal working document a draft passa poco e i draft dopo pochi mesi scadono. Dopo massimo due anni o si rifiuta o si fa il draft standard (draft RFC) che o lo si rifiuta o diventa standard in max 4 anni. Necessita di più implementazioni interoperabili.



0.4 Abstract syntax notation one

ASN1 Sintassi per la definizione di strutture dati e formato di messaggi. Ha l'obiettivo di consentire a macchine dalle differenti architetture hardware di scambiare dati, essere language neutral e consentire la negoziazione della codifica di trasmissione.
Come spostare le informazioni? Vari costruttori all'inizio lo facevano "in casa" senza interoperabilità. Col tempo si è reso necessario costruire qualcosa per scambiare le informazioni in maniera interoperabile.

Endian Come si ordinano i dati in spedizione, per sapere qual è il più significativo.
Bit più significativo a sx è big-endian, ormai poco usato. Altrimenti è little-endian.

0.5 lezione 4

Nel monitorare il traffico di rete c'è il problema di come riceverlo. Non sempre siamo nel posto giusto. Se voglio vedere cosa fa altro dispositivo/sottorete a livello di traffico, come faccio? Opzioni: o possiamo mettere la mano sul pc (wireshark) o posso fare finta di essere il pc (chiedendo allo switch, non intrusivamente, di mandare il traffico verso pc pure a me) Prima di iniziare a guardare il traffico, il traffico va visto.

0.5.1 Common problems with packet capture

...
Perché `root`? Perché scavalco ciò che fa un'applicazione, perché vedo tutto il traffico indipendentemente dall'applicazione. Per questo devo essere root.
Container condivide kernel con host, macchina virtuale emula il kernel.
Necessità vedere traffico. Non vorrei mettere mano sulla macchina, perché devo avere so che permette, utente, installare software...
Quindi lo faccio da fuori, prelevandolo dallo switch.
Metodi software: ho switch, che ha delle porte: port mirror: tutto traffico diretto verso tale macchina oltre a mandarglielo lo mandi anche a me su questa porta. 1:1 una porta verso una porta, 1:N tot porte switch le mandi qua. VLAN mirror: simile al port mirror: dammi tutto traffico tale VLAN e mandalo qua traffic filter/mirroring: dammi solamente traffico di tale porta tale ip...

Singolo cavo ha 2gbps (1gps in upload e 1gbps in dwnld), quindi con port mirror, che posso scaricare al massimo a 1gbps, ho efficacia se traffico sta solo a 1gbps. Dovrei avere scheda di rete da 10gbps per reggere comodamente traffico e non perderlo. Scheda direte più veloce della somma delle due direzioni.
Hardware: Network Tap prende le singole direzioni del traffico e ne fa una copia. PC monitor con due schede di rete perché tap divide il filo le direzioni: una prende le direzioni in entrata e una prende la direzione in uscita. Così scopro anche chi invia cosa. Nel port mirror non sono preservate le direzioni.

0.6 Lezione

Software di switch e infrastruttura di rete devono essere duraturi, perché infrastruttura di rete si cambia quando c'è veramente necessità. Altrimenti infrastruttura rimane lì.

Problema Gestire le cose nel tempo, che rimangano interoperabili negli anni. Siccome informatica va avanti per mode, si sono posti come problema (fatta negli anni '80) dover gestire qualche sistema non attraverso la url come si fa ora, perché è un modo di fare molto volatile che cambia spesso. Allora hanno fatto standard con negoziazione alto livello: oggetto ha attributi ad alto livello, funzionali al suo funzionamento (macchinettà caffè: c'è acqua, quanti caffè fatti, se ha bicchierini...).
Attributi, cosa ti mostro io che sia rilevante per te (non quante viti ha, ma lo stato), operazioni che si possono fare su quegli attributi (accendi, spegni...), comportamento. Software si occupa dell'accensione, non so com'è fatto internamente, io mi limito a chiamare l'operazione. Standardizzazione di funzionamento.

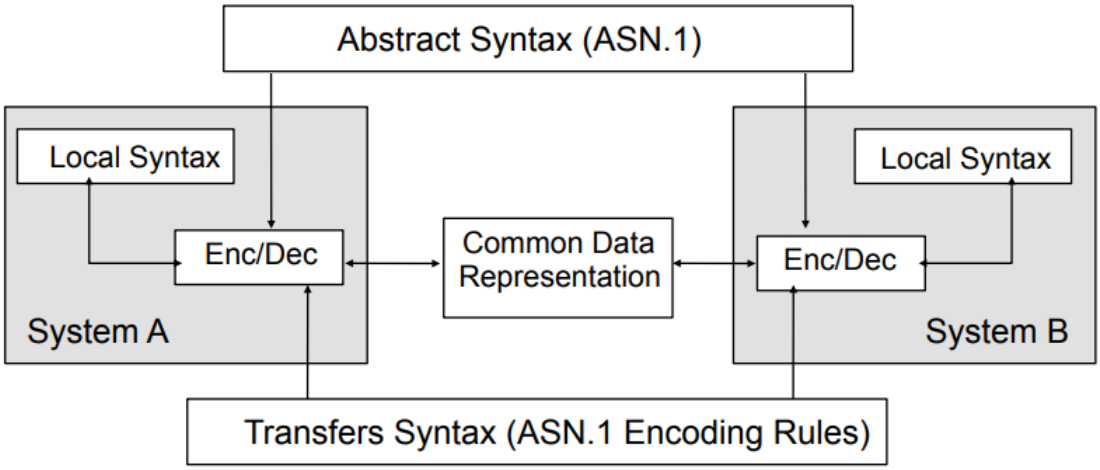
Manager Comanda l'operazione, impone politica di gestione.

Agent Gira dentro la macchina gestita e fa cosa chiede manager e solo quello.

Paradigma Agent gira nella macchina monitorata, e un solo manager raccoglie dati e visualizza.

Come realizzarlo Bisogna negoziare rappresentazione dati. Per lavorare col web hanno risolto problema scambio dati convertendo tutto a stringhe. Questo modo di fare non ha grandi problemi, ma è inefficiente. Grande quantità di dati per poche informazioni (`true` scritto invece di un bit). Poco efficiente per tanti dati, chiaro ma non compatto. La URL è breve. Tra due macchine posso scambiarmi dati in maniera binaria, ma bisogna mettersi d'accordo. Se io a 16 bit parlo con una a 64 bit non ci capiamo, quindi bisogna accordarci (Little Endian e Big Endian).

ASN1 Sintassi astratta, implementata dai linguaggi. Quando iniziano a parlare due macchine negoziano rappresentazione e codifica.



La sintassi locale (**local syntax**) è diversa e tipicamente dipendente dal linguaggio utilizzato: ad esempio una in GO l'altra in C, ma anche per sistema A Arduino Nano e sistema B workstation Windows.
L'ASN1 quindi definisce una sintassi astratta standardizzata. Permette diverse regole di codifica che trasformano la sintassi astratta in un flusso di byte adatto al trasferimento: **BER** (Basic Encoding Rules) definisce il mapping tra sintassi astratta e sintassi di trasferimento.
ASN1 rimane architettura, idea. La sintassi di trasferimento può essere JSON, GO o qualsiasi altra cosa: l'**importante è che le due applicazioni si capiscano**. I **tipi di dato** (datatypes) primitivi dell'ASN1 sono:

`BOOLEAN`

`INTEGER`

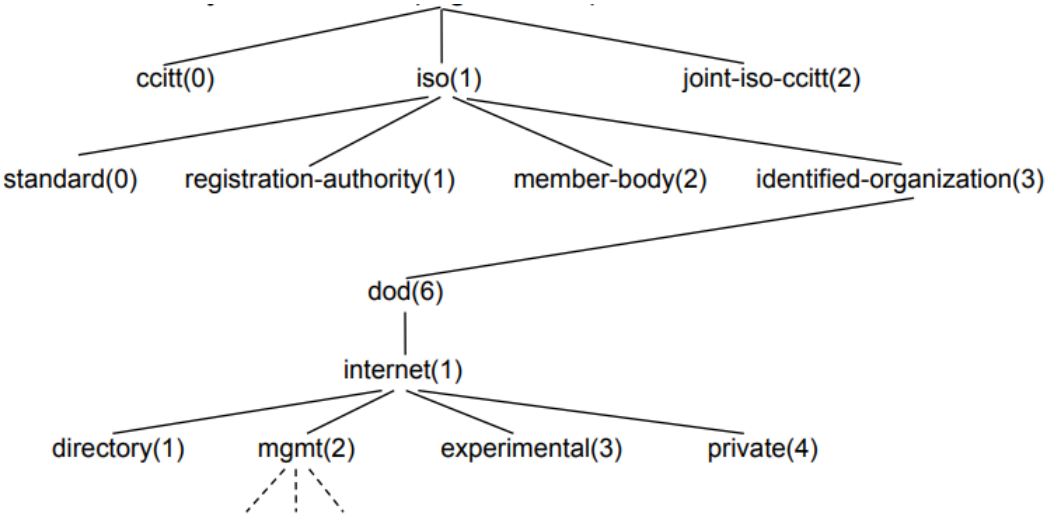
`BIT STRING`

`OCTET STRING`

`OBJECT IDENTIFIER`
Quando trasferisco dati da applicazione ad applicazione, devo poter indicare un campo chiave. Questo tipo di dato identifica univocamente l'oggetto che sto trasferendo all'interno dell'albero ISO

...

ISO Registration Tree Usato per identificare univocamente definizioni, documenti, oggetti... Ha una struttura gerarchica, simile ai file system gerarchici. Tutti i nodi di un livello sono univocamente identificati da un numero. Il **percorso dalla radice al nodo** fornisce una **sequenza numerica** chiamata **Object Identifier**.
Per esempio, Internet è 1.3.6.1



Internet si trova sotto il **Dipartimento della Difesa**, che è una **Organizzazione Identificata** facente parte dell'**ISO**. Noi parleremo della **Gestione** (**m**ana**g**e**m**en**t**) di Internet.
L'**object identifier** quindi, ovvero la sequenza di numeri, risolve il problema dell'identificazione univoca della tipologia dell'oggetto trasmesso sulla rete.

Tipi Complessi ASN1 ha anche tipi complessi, come `SEQUENCE OF` che specifica una lista di dati omogenei, o `REAL` che specifica i numeri reali con mantissa ed esponente dagli `INTEGER`.

Basic Encoding Rules Regole di codifica, compattano i dati in una stringa di byte da spedire sul filo. Basato su un algoritmo tag/length/value (TLV), dove ogni variabile è identificata da un tag, la lunghezza del valore in byte e il valore di quei byte. Questo permette al ricevente di ricostruire il tipo del messaggio a partire dal flusso di byte ricevuto.

Capitolo 1

Gestione di Rete

1.1 Nascita

Gestione di rete nacque, storicamente, nel mondo della telefonia. Necessità di codificare numero, connettersi al centralino, riconoscere il numero.... Standardizzazione necessaria per poter far telefonare a distanze elevate, attraverso le nazioni.

Dati Poi arrivò internet. Prima bisognava instradare la voce, ora vanno istradati i dati (anche voce, `VoIP`, ma pur sempre dati). Eredità di tutte le tecnologie e teorie dei tempi della telefonia (es. 5G) ma anche innovazione (es. Browser Web) ma mantenendo il paradigma che era tutto sommato efficiente.

1.2 Gestione di Rete Internet

Cos'è Sistema di protocolli e tecnologie che permettono di mettere in funzione e controllare un'infrastruttura di rete, per far sì che sia efficiente e che faccia ciò che voglio e che segnali eventuali problemi e comportamenti non previsti.

Reti Geografiche Questo discorso si applica a reti geografiche, ampie e complesse, che interconnettono un elevatissimo numero di device. Serve anche per far sì che la connessione/disconnessione di dispositivi non crei problemi, e che un utente della rete non possa creare disservizi e potenzialmente tirarla giù.

Anni '90 Il problema principale era mantenere bassi i costi, perché doveva essere pervasivo e poter mettere router in ogni casa. Centraline telefoniche, al contrario, non devono stare in ogni casa.
Gli apparati di rete quindi devono costare poco, perciò la gestione di rete non deve costare tanto (nella telefonia costa tanto ed è complicata, quindi "*non facciamo lo stesso errore*": se è semplice anche il costo computazionale è basso, quindi il dispositivo è più economico). Il protocollo, quindi doveva essere **semplice** e **efficiente**.
Altra cosa importante era **l'ubiquità** del protocollo: doveva essere disponibile su tutti i dispositivi, così da poterli gestire tutti.
Inoltre il protocollo doveva essere **estensibile**. Almeno **retrocompatibile**.

1.3 SNMP

Piccoli passi Il protocollo SNMP è stato progettato di pari passo con il diffondersi di internet, prima a livello universitario e poi industriale. Si è iniziato a sviluppare questo protocollo di gestione dagli albori, perché sin da subito è apparso chiaramente l'importanza che l'infrastruttura stia in piedi.
L'SNMP monitora lo stato della rete, per far si che la rete risponda alle esigenze. Ci sono più standard, con primi sviluppati nel 1990.

Semplice Doveva essere **semplice**, poiché i sistemi erano semplici, poco potenti e a volte nemmeno multitasking. L'SNMP non poteva girare "in hardware", ma **necessita di un computer perché necessita di elaborazione dati** e dello stack IP per comunicare. Negli anni '90 lo stack IP non era necessariamente presente sui computer in commercio.
La parte importante è che sia **semplice** e funzionare sotto l'UDP, che è un protocollo estremamente semplice.

Separato La parte dell'SNMP **è separata dalla parte di comunicazione** anche se aiuta l'instradamento. **Non interferisce**, come il cruscotto della macchina (SNMP) col motore (switching).

Trasparente Sta fuori dalla comunicazione, ma **deve poterla controllare** e monitorare **senza interferire**. L'idea è che se l'SNMP viene compromesso lo switch continua a funzionare.

Evoluzione Nel 1990 viene standardizzata una versione molto semplice: l'**SNMPv1**. Questa versione fu prodotta in fretta, concentrandosi sulle funzionalità base, in modo da poter entrare subito sul mercato che stava per esplodere. Nle 1991 viene pubblicata la **Management Information Base**, ovvero l'insieme degli oggetti manipolati tramite l'SNMP.
Negli anni successivi ci sono varie evoluzioni del protocollo:

SNMPv1 supportata da tutti i dispositivi sul mercato

SNMPv2 aggiunge poche funzionalità, ma è molto usata soprattutto perché i contatori ora sono a 64bit

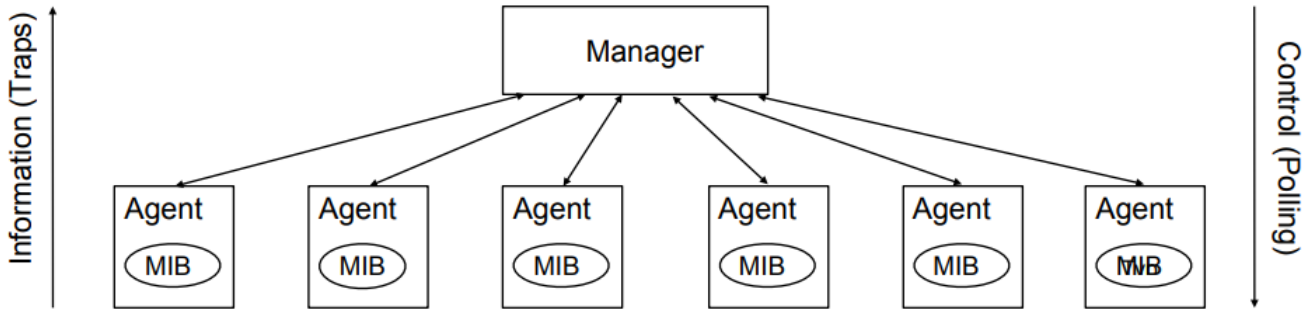
SNMPv3 aggiunge parecchie funzioni, sacrificando il "**simple**", quindi non è particolarmente diffuso

Utile L'SNMP è quindi utile per il monitoring centralizzato su reti estese: è **importante che questi protocolli siano in funzione in ogni momento**, per riconoscere i problemi in anticipo e avere uno storico della rete per poter fare le verifiche. Anche solo contare il traffico prodotto in termini di byte è un'informazione molto importante.

Agent **Apparato di rete**, ad es. nella rete del Fibonacci ci sono diversi agent: access point, computer, stampanti...
Sono gli **oggetti da gestire** e possono cambiare nel tempo: possono essere aggiunti/rimossi, ma possono anche cambiare in tipologia

1.3.1 Trap Directed Polling

Tutti gli agent rispondono ad un manager (solitamente ridondato) $\rightarrow n$ manager per un solo agent.



Uguaglianza Con lo stesso protocollo e la stessa funzionalità, devo **poter controllare dispositivi anche molto diversi**: computer, stampanti, badge, schermi...
L'SNMP **non distingue i dispositivi**, ma **prende le info dalla MIB** del singolo agent. Ciò che differenzia i vari sistemi operativi, i dispositivi tra loro ecc. sta tutto nella **MIB** (Management Information Base).

Polling Controllo. Il manager non comunica continuamente con gli agent, ma esegue un **polling degli agent**, contattandoli periodicamente per **ricevere le informazioni aggiornate**.
L'SNMP è altamente centralizzato, quindi è compito del manager implementare tutta la funzionalità di monitori e la responsabilità, sicurezza ecc...

Traps Una volta configurata, sta alla **periferica avverte il manager se qualcosa non sta funzionando come previsto**. Non informa il manager ogni volta che succede qualcosa (stampo un foglio, prendo un caffè, mi connetto all'Access Point...) perché il manager verrebbe inondato di informazioni, ma **l'apparato informa il manager se ci sono cose che non funzionano**. Segue la filosofia del "*Se non mandi niente va tutto bene*".
Questo può non essere sufficiente, per esempio in caso di problema di rete la comunicazione può non andare a buon fine. In tal caso il polling può risolvere questa cosa, anche se ciò significa apprendere il verificarsi del problema in maniera non tempestiva.

SMI La struttura delle informazioni di gestione (**Structure of Management Informativo**, la seconda versione) si basa su un sottoinsieme dei datatype ASN.1 più altri sottotipi:

`Integer32` interi con il segno

`Unsigned32` interi senza segno

`Gauge32` per misure pronte comprese tra min e max

`Counter32` e `Counter64` per i contatori: per conoscere la misura effettiva devo fare la differenza tra il valore assunto in due istanti separati.

`IpAddress` per IPv4

`TimeTicks` per i centesimi di secondo passati

`Opaque` che è come il `void`, non consigliato

Le variabili, inoltre, possono essere

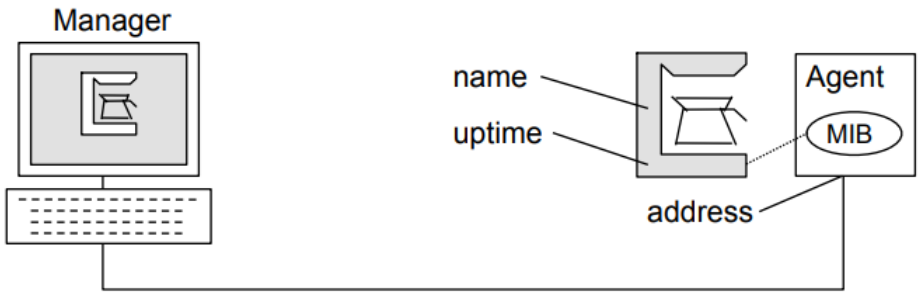
Scalari: esistono una sola volta per agent. Es: il nome di un host.

Concettuali: esistono in una tabella concettuale, con valori che cambiano nel tempo.

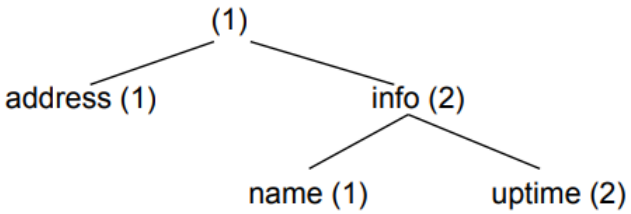
Read o **write**, lette o scritte. Non esistono incrementi o reset a valori iniziali, questo per semplicità di protocollo

MIB Le MIB di SMIv2 sono **definite tramite speciali macro ASN.1**

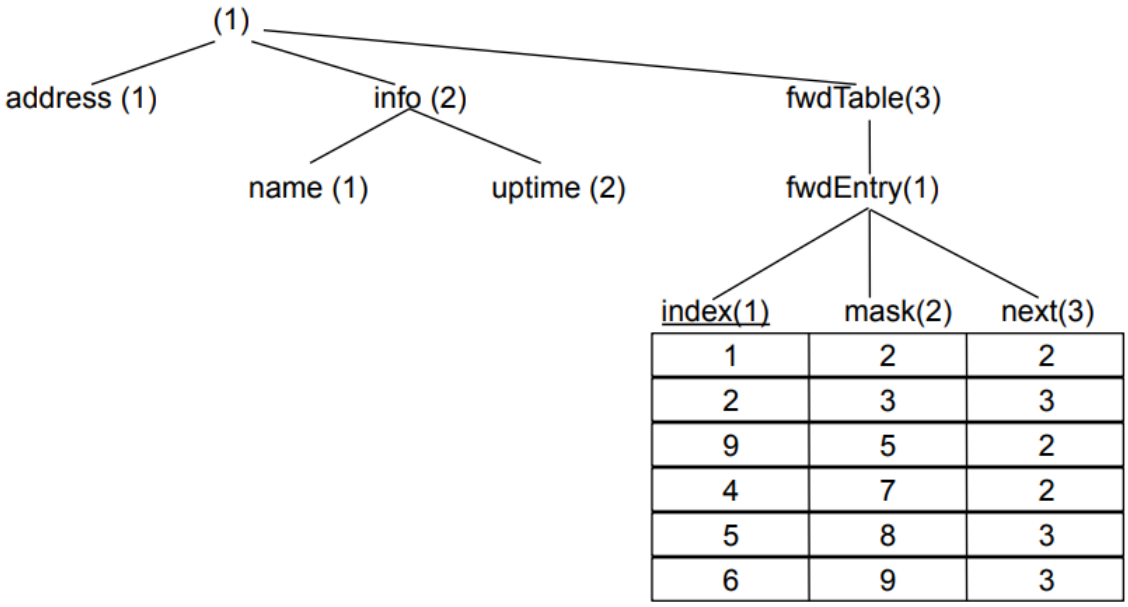
Use Case



Le variabili sono definite nell'ISO Registration Tree. Una foglia dell'albero rappresente on managed object.



Instance Identifier Ogni managed object è identificato dall'object identifier. Ma tale oggetto ha istanze con determinati valori che cambiano nel tempo: tale istanza è identificata univocamente da un **Instance Identifier**. La singola istanza di un managed object è **univocamente identificata concatenando l'Instance Identifier all'Object Identifier**.
Gli scalari hanno una sola istanza, e l'istance identifier è `.0`. Negli altri casi (non identifier) si parte da `.1`.
L'istance identifier è il valore della colonna indice.



index(1)	mask(2)	next(3)
1	2	2
2	3	3
9	5	2
4	7	2
5	8	3
6	9	3

In questo caso, per prendere il valore della terza istanza di `mask`: `1.3.1.2.9` → `5`.

Altri esempi:

`1.3.1.1.1` → `1`

`1.3.1.2.1` → `2`

`1.3.1.3.1` → `2`

`1.3.1.1.4` → `4`

`1.3.1.2.4` → `7`

`1.3.1.2.7` → ⍰

1.3.2 MIB

Come si definisce un MIB

```
nome DEFINITIONS ::= BEGIN

IMPORT MODULE-IDENTITY, OBJECT-TYPE, enterprises, IpAddress, TimeTicks FROM SNMPv2-SMI;
...

END
```

I nomi servono solo per gli umani, perché sulla rete passano gli object identifier.

Module Identity Non aggiunge informazione ma indica la versione, questo perché negli anni le reti cambiano ma l'infrastruttura rimane fissa almeno in parte. I device a lungo termine (ad esempio lettori di badge) devono convivere con i nuovi: eterogeneità nelle velocità di elaborazione, protocolli...
Non posso togliere oggetto, perché magari versioni precedenti lo usano. Se voglio eliminarlo posso indicarne lo `STATUS`: obsolete, current o deprecated.

Object Type `nome OBJECT-TYPE, SYNTAX, UNITS...`

Notification Type Invio anche oggetti interessati alle trap
linkDown per segnalare se ha perso il link, linkUp per collegamento preso

Utilità dei MIB Compilatore non è detto che generi linguaggio macchina ma trasforma semplicemente. I **frontend compilers** SNMP prendono i MIB e producono warning/errori e conversioni di formato SMI Quando è nato snmp fatti due standard: snmp e mib iniziale (mib-ii, rfc 1213) dove vengono descritti object più importanti per gestione internet.
Vogliono controllare stack TCP/IP per vedere cosa non funziona poi monitorare porta di rete. device espongono contatori per interfaccia tramite snmp. contatori byte pacchetti sono base del monitoraggio, ma nel tempo và superato

perché contatori su byte sono poco utili: non si sa bene cosa succede però.
Obiettivi mib-ii: info base su errori, pochi e semplici control objects (su/giù, errori, pacchett/byte ingresso/uscita), cerca il più possibile di evitare info ridondanti, **non deve assolutamente interferire con le operazioni** devono essere separati e non rallentare o inficiare in nessuna maniera prestazioni dei dispositivi. Sono circa 170 oggetti, negli anni alcune def sono troppo semplici, e presuppone IPv4 (IPv6 gestito in mib separati).

schema dei MIB-II (1.3.6.1.2)

Composizione Diviso in gruppi: ...
La parte trasmissione è aperta, tanti sottoalberi: com'è connesso computer dipende dalla tecnologia... quindi è estendibile a seconda delle nuove tecnologie
Quindi negli anni viene modificato ma è integro sin dalla nascita.
Per info più specifiche ci sono mib successivi.

tabella

Statistiche su stessa rifa sono a livello diverso, repeater a livello 1. Fanno stessa cosa ma livelli diversi.
Per navigare tabelle necessità meccanismo efficiente rispetto a fare tante richieste. Questo perché sistemi modulari, per vedere se valore/elemento esiste.

Ordine lessicografico Walk:lettura consecutiva di oggetti in mib.
Leggo oggetti e metto ordinati per numero in ordine lessicografico. Con questo ordinamento si perde la struttura della tabella: **SNMP lavora solo su questo array ordinato**.

Trap Maniera asincrona per informare manager di qualche avvenimento, tipicamente un cambio di stato.
`ifAdminStatus` su `linkDown` equivale proprio a scollegare fisicamente la scheda di rete. L'**administrative** è il ciclo di vita, l'`operational` è operativa. `ifOperStatus` su `linkDown` è lo stato e significa che la connessione è caduta.

1.3.3 Primitive

Due tipi di valori, scalari e tabellari.

Get Richiesta diretta e precisa di un object identifier
Può essere usato per leggere una o più variabili. Basato su UDP, quindi dimensione massima pacchetto: nella risposta bisogna stare attenti alle info. Possibile errore (**errorStatus)**: tooBig, non entra la risposta nel pacchetto UDP. Altri errori: noSuchName (istanza non esiste o non è foglia, esempio chiedere bicchierini ad una stampante), genErr (qualsiasi altro errore: comunità sbagliata, agent sovraccarico...).
errorIndex indica quale delle variabili ha avuto problema. Es `noSuchName@1` indica che la prima variabile non esiste. Posso fare get con più richieste, quindi errore può essere su altra variabile (es `noSuchName@3` cioè non esiste il terzo object identifier richiesto ma indica il primo indice fallito, quindi le altre due precedenti esistono ma non vengono ritornati i valori).

GetNext Richiesta che ritorna l'object identifier successivo a quello chiesto.
Non legge istance identifier richiesto ma ritorna il prossimo istance identifier rispetto all'ordine lessicografico. usata per fare discovery delle strutture e leggere le tabelle.
Nella getNext, nosuchname significa che è finito il MIB (non esiste il successivo). Molto implementation depended, alcuni dopo l'ultimo ritornano il primo invece che noSuchName.

Set Scrivere un valore
Equivalente in sostanza alla get ma scrivo. Atomica: più valori sono scritti contemporanemante, quindi o vanno bene tutti o non scrivo niente.
Errore **badValue** quando valore scritto è del tipo sbagliato, fuori dal range o comunque non accettabile. Ma anche errore quando oggetto è `read-only`.
noError quando tutto ok, noSuchName quando instance identifier non esiste. Anche qua stesso discorso della get con l'indice.

Trap Trap dall'agent
Unico messaggio non richiesto che va dall'agent verso il manager.
Può succedere trap storm, esempio dopo perdita alimentazione tutti dispositivi segnalano riavvio apparati tutti insieme. Se tolgo corrente non posso fisicamente mandare trap, per questo succede quando riavvio e torna su, non quando va giù tutto insieme.
Non ci si può fidare totalmente della trap, bisogna fare polling.
`ColdStart/WarmStart` inviate quando avvio un agent (cold da freddo, spento, caso tipico, invece warm è da riavvio)
`LinkDown/LinkUp`
`AuthenticationFailure` quando faccio richiesta posso non avere autorizzazioni, trap segnala al manager della situazione (tipicamente se provo tante password una dietro l'altra).
`EnterpriseSpecific`, vedi valore enterprise e specific nel formato del pacchetto. ad esempio fine carta di una stampante. Specific=1 per dire che l'interpretazione è a seconda del mib.

Formato pacchetti

Community: una sorta di password, vedo se utente che richiede è abilitato a certa operazione. PDU payload data unity
Get getnext e set ricevono getresponse tutte e tre.
Importante sapere IP host che manda trap (potrebbe vederlo dal pacchetto UDP), perché l'ip nell'udp può essere mascherato o cambiato ad esempio dal nat

`sysDescr`(1) breve descrizione del sistema (stringa, **arbitraria**)

`sysObjectID`(2) identifica sia modello che costruttore, per riconoscerlo e andare nel relativo MIB o, ad esempio, mettere l'icona giusta.

`sysUptIme`(3) da quanto è attivo l'**agent** (non è per forza identico a quello di sistema). Serve anche per poter leggere correttamente le variabili `gauge` per esempio, per fare la differenza per sapere quanto è cambiato il valore nell'ultimo minuto (ad esempio). Ma non posso aspettarmi che il nuovo valore sia maggiore (es: è saltata la corrente, oppure se il contatore ha fatto wrap (fine dei bit e ripartito da 0)) Quindi devo distinguere riavvio agent da wrap, non posso fare semplice differenza perché potrei tirare fuori numero sbagliato (es negativo, ma valore `unsigned` fa diventare grandissimo). Per sapere se si è riavviato agent o se contatore ha fatto wrap uso `sysUpTime`.

1.3.4 SNMPv2

Usato snmpv2 invece che v1 perché aveva problemi strutturali: il primo sono i contatori a 32bit che sono molto limitati per la tecnologia odierna (10-100Gbit il contatore si "riempie" molto velocemente). Inoltre nella v2 c'è tutto quello scartato in v1 per questioni di tempo (necesario fare protocollo in fretta)

Primitive di SNMPv2 Sono tutto sommato le stesse con `GetBulk` e `Inform`

> `GetBulk` per richieste più efficienti per massimizzare numero richieste in un pacchetto. Metà strada tra `Get` e `GetNext`. Oltre OID due parametri:
>
> > `non-repeaters`: di tutti gli OID specificati, `non-repeaters` indicano quanti di quelli sono quelle variabili a cui applicare una `Get` secca e non in sequenza. Dei restanti, devo fare `max-repetitions GetNext`
> >
> > `max-repetitions` quante `GetNext` applicare
>
> Se chiedo poche cose non risparmio molto rispetto ad una `GetNext`, se chiedo troppe cose (`max-repetitions` troppo alto) devo stare attento che entri tutto in un pacchetto.
>
> `Inform` è una sorta di `Trap` informata perché riceve una `response`. Altra differenza è che la `Inform` la può mandare sia un Manager (che può informare un Manager di livello superiore, per esempio) che un Agent

Formato Hanno la stessa PDU ma con modifiche all'interno, però sostanzialmente è uguale

Eccezioni A differenza degli errori di SNMPv1 sono più esplicativi. Ad esempio `noSuchName` spacchettata in `noSuchObject`, `noSuchInstance` o `endOfMibView`. Sono un **raffinamento dei codici di errore di SNMPv1** e un **miglioramento delle primitive**.

Differenze principali Più velocità con la Bulk, contatori a 64bit, errori molto più dettagliati. Ma **non risolve granché** in termini di sicurezza.

1.3.5 SNMPv3

Obiettivi Bisogna risolvere problemi di sicurezza soprattutto per la `Set`. Inoltre bisogna fare il design di un'architettura a lungo termine.
Supportare implementazioni sia complesse che semplici (**scalabilità**).
Il tutto **rimanendo il più semplici possibile**. Nella realtà non è stato così, e la diffusione è ancora molto limitata con la maggior parte delle reti che continuano ad usare v1 e v2.

Architettura Tanti componenti, rimovibili e intercambiabili. Vuoti se una certa implementazione non la supporta, rimovibili a seconda di cambiamenti tecnologici o di implementazione.
Supporta vari tipi di protocollo per lo scambio di messaggi e per il sistema di sicurezza (comunità, utente, "*altro*")

Caratteristiche più importanti

1. **Sicurezza**
   Il messaggio è autentico? Data Integrity e Authentication
   Tramite una funzione hash, con chiave simmetrica faccio l'hash dei dati e calcolo un MAC (Message Autethication Code) es MD5. Pacchetto diventa User (key), MAC e Data.
   Il ricevente ricalcola l'hash e se MAC calcolato e ricevuto sono uguali allora apposto.
   Il problema è la **chiave simmetrica**.
2. **Protezione dalla ripetizione di vecchi messaggi**
   Il ricevente conosce l'orario del messaggio ricevuto, se il messaggio ricevuto è nell'intervallo di validità e *più giovane* dell'ultimo messaggio valido, allora è processato e l'orologio viene aggiornato. Quindi, ad inizio comunicazione gli orologi vengono sincronizzati.
3. **Protezione dallo sniffing**: si crittano i dati

Capitolo 2

Rete Cellulare

GTPC realizza autenticazione, il telefono si annuncia sulla rete (register) e disannuncia quando esce, oppure update quando cambia cella. L'annuncia comprende il MEI (identificativo univoco del telefono), IMSI (international mobile subscriber identity) e altre info, ad esempio sul profilo tariffario (velocità max ad es). Sulla rete passano anche info su nodo di rete e identificativo della cella.
Create session crea la connessione e la stabilisce.
Protocollo GTP inoltre negozia i **tunnel**: permettono che cambiando cella cambiano i parametri della connessione ma la comunicazione rimane. Pacchetti incapsulati in tunnel GTP, io continuo a vedere il mio ip verso l'esterno e percepisco come se fosse rete fissa.

Traffico è incapsulato dentro il pacchetto GTP.

Capitolo 3

RRDtool

3.1 RRDtool Database

I db relaz hanno tabelle connesse con relazioni... con inserimento, cancellazione e aggiornamento. Questo non è abbastanza. Inoltre su uso questo modo la tabella cresce all'infinito, durante il monitoraggio
Poi DB relaz non efficienti nel fare aggregazione nel tempo dei valori. Inoltre, i dati più vecchi diventano via via meno importanti.

RRDtool Padre di tanti DB a sede temporale. RR=Round Robin. RRD è un **file**, in un formato particolare:

Static Header che indica cosa contiene, ultimo timestamp di aggiunta e ultimo dato aggiunto...

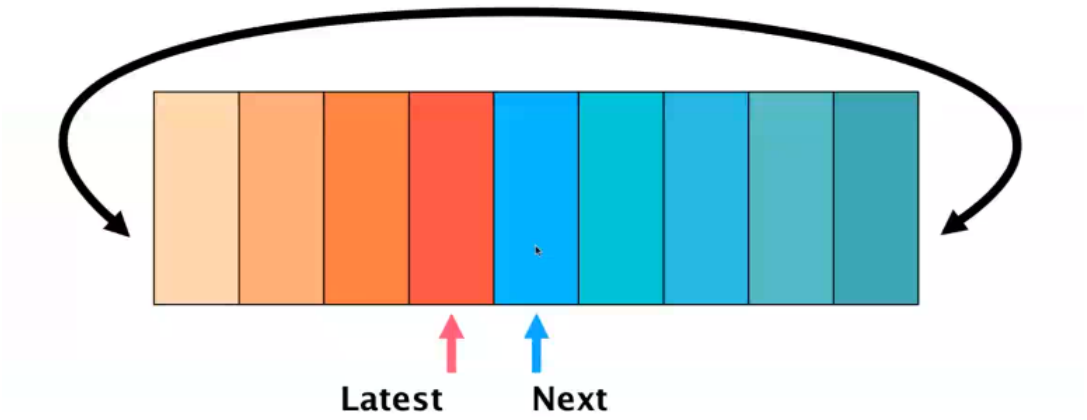
Live Header

Round Robin Archive

Round Robin Archive

...altri RRA

RRD Array circolare con numero fisso di slot di storage.



Data Source Qualsiasi cosa con numeri.

Sconosciuto Come gestire `UNKNOWN`? Unknown non è 0, è contagioso (1+unknown = unknown). RRDtool gestisce gli unknown, configurabile quanto unknown ignorare (default: 50%)

Archivi multipli Tieni i dati pronti alle giuste risoluzioni
Media su 5 minuti per un giorno. Massimo su 1h per un mese

Accorgersi problemi Per accorgersi di problemi, soglie: colorare grafici in base a soglie (sopra 60 giallo, sopra 85 rosso ad esempio). Però sono statici.
Altrimenti in base ai valori storici: vedo qual'è la media nel passato e se mi discosto tanto...
Una cosa interessante che fa RRDtool. La derivata da indicazione di quanto mi discosto dall'osservazione precedente. Quindi la prima domanda è quanto tempo osservare. Anche dalla natura della metrica: una derivata violenta significa problemi al sensore.
Inoltre la derivata su altre misure possono anche essere molto ampie senza che sia allarmante: download/upload, ad esempio.
Però il color-coding: verde ok, giallo mh e rosso male male, è ottimo. Keep it simple
Holt-Winters.

tutti valori letti dalla scheda di rete

ifindex: identifica l'interfaccia

ifdescr: descrizione interfaccia (ad es: eth0 ma anche descrizioni significative)

iftype: tipo interfaccia (ethernet, ...)

ifmtu: maximum transfer unit, indicata per poter evitare cattive configurazioni nell'mtu. MTU deve essere la stessa perché se uno trasmette a > mtu di uno dei due non comunicano più

ifspeed:

ifphysaddress: indirizzo fisico della porta di rete, mac address della porta dello switch

ifadminstatus: se scheda è fisicamente presente sullo switch

ifoperstatus: per una scheda presente, se è operativa oppure no

iflastchanged: sysuptime dell'ultima volta che l'interfaccia è cambiata di stato

ifinoctets: otteti in

ifinucastpkts: unicast in

ifinnucastpkts: not-unicast in

ifindiscards: discarded in

ifinerrors: errori in

ifinunknownprotos: sconosciuti in

ifoutoctets: otteti out

ifoutucastpkts: unicast out

ifoutnucastpkts: not-unicast out

ifoutdiscards: discarded out, dipende dall'implementazione della scheda

ifouterrors: errori out, dipende dall'implementazione della scheda

ifoutqlen: stessa info del comando `netstat` e simili, info sulle code in uscita

ifspecific:

router frammenta il pacchetto quando mtu della linea ricevente è più piccolo. ma può essere che mittente mandi pacchetti frammentati quando qualcuno chiede a lui di mandare pacchetto troppo lungo rispetto mtu.
Frammentazione è concetto dell'IPv4
TCP frammentazione non c'è ma non perché non si possa fare, ma perché un pacchetto TCP non può essere frammentato (è indicazione di errore che qualcosa nel TCP non ha funzionato). Path MTU discovery (PMTUD), pacchetti sempre più grandi finché non si trova dimensione massima con l'**obiettivo di evitare la frammentazione**. Perché meccanismi TCP permettono di farlo. UDP pacchetti vivono di vita propria quindi può succedere frammentazione.
Pacchetti non unicast sono "problematici" perché riempino le porte dello switch.

Capitolo 4

Bridge MIB

Bridge è mettere assieme due interfacce di rete
Risponde alla domanda: come fare per iniziare a vedere la topologia della rete e, quindi, i dispositivi collegati ad un host?
Attraverso bridge-mib per la topologia di rete. Importante anche per inquadrare che impatto può avere un'allarme sul sistema. Così so sia quali sono i nodi principali ma anche l'impatto di qualche problema: se cade una macchina che collega tante altre macchine, cadono tutte, mentre se cade no nodo marginale il problema è più contenuto.

Capitolo 5

Cattura

Quando cattura traffico intervengo sul traffico e me lo copio verso l'applicazione che cattura ciò che sarebbe destinato ad altri. Quindi ethernet copia pacchetto al BPF driver che, tramite filtri, manda copie agli sniffer.
Gli sniffer necessiterebbero di diritti su

Capabilities Quando uno è SU ha diritti eccessivi rispetto a ciò che deve fare, quindi le **capabilities** sono le possibilità di spezzare i diritti di amministrazione in categorie: non c'è bisogno di essere root che fa veramente veramente tutto perché se ne può abusare. Quindi il concetto di root è spacchettato in varie possibilità: configurazione del MAC address, Net Admin...
Un'app quindi dovrebbe nascere con conetto di capabilities e avere diritti minimi necessari per fare ciò che vuole/deve fare.

Filtro Devo indicare l'interfaccia a cui sono interessato catturare pacchetti (eventualmente tutte) e in caso che tipo di pacchetti ricevere. Così il BPF driver **non** invia all'applicazioni pacchetti non interessanti, perché anche scartare qualcosa richiede tempo.
BPF e sottosistema è localizzato nel kernel. Solamente la parte del pacchetto è inoltrata, il resto (filtro e copia dei pacchetti) è nel kernel.
Nel mondo linux, la cattura dei pacchetti equivale ad aprire un socket.

5.1 `libpcap`

`pcap_open_live` Parametri: `deviceName` scheda di rete, `maxCaptureLen` lunghezza massima del pacchetto (sia per efficienza, sia per privacy se non ho bisogno dell'intero pacchetto), `setPromiscuosMode` per abilitare la modalità promiscua (cattura tutto il traffico sulla scheda), `pktDelay` non c'erano i thread quindi hanno pensato di mettere un tempo massimo per cui aspettare un pacchetto e in caso ritornare senza valore, `errorBuffer` sempre per ragioni storiche se la `pcap_open_live` fallisce il buffer contiene la ragione per cui è fallita
`live`: leggo traffico di rete in tempo reale. C'è la possibilità di aprire file `pcap` per leggere traffico salvato su file.
In realtime bisogna controllare il tempo del pacchetto con il mio tempo. L'orario contenuto nel pacchetto è quello della sua ricezione, quindi nell'rrd è registrare i dati all'orario del pacchetto, non a quello di sistema

`pcap_compile` Compila l'espressione del filtro

`pcap_setfilter` Trasferisce nel kernel l'espressione compilata

`pcap_next` Prende il prossimo pacchetto catturato

Capitolo 6

Monitoring

monitoring requirements: non è chiaro cosa fare alla rilevazione, ma è chiaro che bisogna fare qualcosa.

Capitolo 7

Benchmarking per dispositivi interconnessi

`RFC 1944` Definisce come eseguire misure del traffico di rete:

- Architettura per il test (dove piazzare il sistema testato)
- Dimensione dei pacchetti usati per le misure
- Indirizzi IP da assegnare al SUT (**System Under Test**)
- Protocolli IP da usare per il test (es: UDP vs TCP)
- Uso di picchi di traffico durante la misura (picchi vs traffico costante)

Definisce e specifica come:

- Verificare e valutare i risultati dei test
- Misurare metriche comuni definite nella `RFC 1242` come throughput, latenza, frame loss
- Gestire "test modifiers" come
  - Traffico di Broadcast
  - Durata del test

Metriche

Disponibilità La disponibilità è espressa come la **percentuale del tempo che un servizio è disponibile**. È la prima misura di affidabilità di un sistema. Si basa sull'affidabilità del componente di rete individuale.

$$\%disponibilità = \frac{MTBF}{MTBF + MTTR}$$

MTBF = mean time between failures
MTTR = mean time to repair following a failure

Tempo di risposta Quanto tempo ci mette il sistema a reagire ad una richiesta

Accuratezza Quanto posso essere accurato nel fare la misura?

Throughput

Utilisation

Latency e jitter

ETSI Definisce metriche che nel mondo internet non sono state definite.

Capitolo 8

Deep Packet Inspection

Traffic classification Traffic classification importante per capire cosa viaggia sulla rete. SNMP permette vedere se unicast o meno, ma non dice niente del traffico: buono, conosciuto...
Per classificare traffico quattro metodi:

1. Basato sulle porte TCP/UDP
   Più o meno sempre fatta, specialmente nei primi anni di internet. Identificati da protocollo e porta nel range delle well known ports. Facile da evitare (porte dinamiche) quinidi inaffidabile (TCP/80 $\neq$ HTTP)

2. Flag del pacchetto (DSCP)

3. classificazione statistica
   Di moda per un certo periodo, utilizzo del machine learning per classificare pacchetti. Si pensava che ML costasse meno rispetto analisi profonda del pacchetto

4. Deep Packet Inspection
   Analisi del pacchetto internamente. Selective metadata extraction (HTTP URL o User-Agent) necessario per fare monitoring accurato. Lo fa il DPI toolkit senza replicarlo sulle applicazioni di monitoring.

Protocolli in nDPI Identificati da `major.minor`
Major è protocollo di rete, minor è protocollo.
Oggi molti protocolli basati su HTTP e TLS. nDPI supporta riconoscimento protocolli basato su stringhe: DNS query name, HTTP campi host/server, SSL/QUIC SNI
I protocolli sono tantissimi, ma nDPI consente di raggrupparli in categorie, che possono includere migliaia di protocolli ed essere (ri)caricate dinamicamente.

Come usare Applicazione cattura il pacchetto e mantiene il flusso di stato, nDPI si aspetta di ricevere il pacchetto. Ogni dissettore è codificato in un differente `.c`, ognuno classifica il singolo protocollo, per modularità ed estensibilità.

Traffic Classification Lifecycle Basato sul tipo di traffico, dissectors applicati sequenzialmente iniziando con quello che più probabilmente matcherà (es HTTP dissector se traffico su TCP/80)
Ogni flusso mantiene lo stato per dissector non-matching per saltarli in futuro.
Analisi dura fino a match o dopo troppi tentativi (8 pacchetti solitamente)
Flusso è bidirezionale e corrisponde alla quintupletta IP+prot+porta+src+dest(+vlan)...
Riconoscimento avviene solamente ad inizio del flusso.
nDPI funziona con i pacchetti IP.