Architettura per la correlazione di trap SNMP provenienti da probe distribuiti

Francesco Galli
Diploma Universitario in Informatica
Università di Pisa
wereboar@interfree.it

Sommario

1. La problematica

Uno dei problemi più sentiti dall'utenza - e quindi importanti - nella realizzazione e nell'uso dei sistemi di management su reti distribuite è la generazione degli allarmi sulla console di management; si desidera infatti un sistema che generi tutti gli allarmi necessari, senza tuttavia "tralasciare" di avvertire l'operatore su condizioni anomale che devono essere segnalate, e senza sommergerlo con una marea di allarmi duplicati.
Gli strumenti più usati nel campo del Network Management sono i più semplici, e la loro semplicità porta con sé una certa "ingenuità"; mancano assolutamente controlli di natura logica sul loro funzionamento, e la loro natura intrinsecamente "meccanica" li rende talvolta inadatti – quando non addirittura fuorvianti – nell'analisi di sistemi complessi come quelli che si sente il bisogno di gestire.

La qualità e l'utilità di un sistema di management non è solo legata quindi alla sua effettiva bontà, parlando in termini canonici di efficienza nell'utilizzo delle risorse o di velocità, ma anche e soprattutto nella sua capacità di relazionarsi con l'individuo che sta sempre a monte della console di management, ossia il network manager stesso, quindi alla *percezione* che l'utente ha della qualità di un sistema. La capacità dell'essere umano di raccogliere, memorizzare e elaborare informazioni a breve termine è limitata; in tale situazione un sistema che sommerge l'operatore di allarmi inutili, ridondanti o eccessivi è destinato ad essere visto come inefficiente, perché non aiuta l'operatore a svolgere il proprio compito. D'altro canto, un sistema che per non sovraccaricare la componente umana di una risorsa di network management eviti di mandare allarmi che possono essere necessari è senz'altro inutile per definizione.

Scopo di questa proposta è di definire uno scheletro di architettura utilizzabile nel maggior numero possibile di situazioni di network management, presupponendo quindi strumenti di facile utilizzo, installazione e distribuzione, evitando al tempo stesso di scendere eccessivamente nel dettaglio, per risolvere il problema della generazione di un numero corretto di allarmi e limitare il carico di lavoro imposto al network manager, senza perdita di informazione rilevante.

Il concetto di "informazione rilevante" è uno dei punti cardine della teoria del network management, inteso come la definizione di metriche e treshold, ed è strettamente correlato con la singola istanza di una situazione di rete gestita; tralascerò quindi nella trattazione una definizione di "informazione rilevante", lasciando al network manager la definizione di metriche e correlazioni che siano significative per la singola situazione in cui si trovi a lavorare.

Nel paragrafo 2 si delineano alcune ipotesi che permettano di semplificare la trattazione quel tanto che basti a non renderla troppo astratta e diluita; nel paragrafo 3 si presenta la struttura dell'applicazione sulla base del paragrafo 2, mentre il paragrafo 4 è dedicato alla discussioni di alcuni "case studies" – problematiche di management che si possano definire "tipiche" – nell'ipotesi di presenza di una applicazione rispondente alle specifiche del paragrafo 3. Il paragrafo 5 è dedicato alla trattazione di eventuali sviluppi, punti deboli e modifiche che si possono apportare alla struttura.

2. Ipotesi: paradigmi, strumenti, tools

Nell'intento di mantenere quanto esposto il più possibile context-independant, presuppongo nella mia trattazione l'uso di strumenti che siano tra i più usati nel campo del network management, a disposizione di tutti e adatti ad ogni contesto.
Si suppone quindi:

1. di lavorare su una rete generica, sia essa concentrata in una unica zona fisica (es. LAN) o distribuita in più punti nel territorio (sia una WAN che una inter-net)
2. Che sulla rete, o sulla singola sottorete, sia installato un protocollo che permetta le operazioni di management di base (lettura, scrittura, e allarme); per semplificare le cose, si suppone una rete TCP/IP popolata di manager e agent che parlino SNMPv1;
3. Che sulla rete, o singola sottorete, siano presenti uno o più generatori di trap "stupidi", vale a dire non-programmabili; per semplicità e coerenza con l'ipotesi 2 ho supposto la presenza di probes RMON-1.

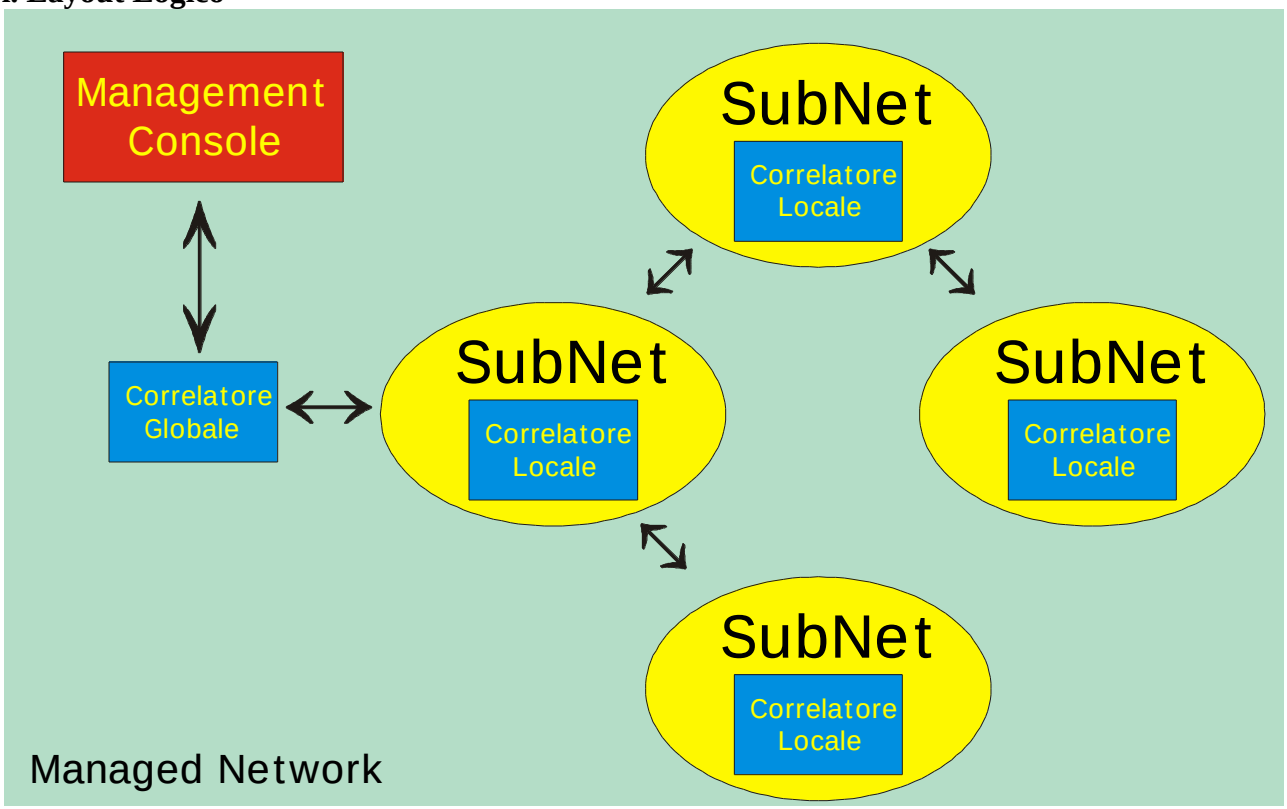
La trattazione fa quindi uso del paradigma "classico" manager-agent, realizzando una struttura di "management by delegation" piramidale, semiautomatica, gerarchica, selettiva e con funzioni di autoapprendimento.
Nella struttura si fa uso di databases; un qualsiasi motore di database, sia esso commerciale, freeware o proprietario può essere adatto, e la sua scelta dipende dalla disposizione della singola situazione da gestire e dalle risorse a disposizione (CPU power, soldi, spazio disco, tempo uomo). Per semplificare le cose, ho supposto un motore di database "generico" che parli SQL-92.

3. L'architettura

a. La Sovrastruttura Totale

Il sistema realizza la correlazione di trap provenienti da ogni tipo di forme, dai generatori "stupidi" di trap, come probes RMON, a generatori "intelligenti" e programmabili di trap, come manager, passando per qualsiasi tipo di agent SNMP configurato per inviare trap autonomamente.

i. Layout Logico



Nella figura un esempio di layout dell'applicazione. Nella totalità della rete gestita è presente – come di consueto – la console di management.
Il correlatore globale si posiziona immediatamente a valle di essa.
Per ogni SubNet – in questo caso intese come LAN elementi di una WAN più grande – è presente un correlatore locale. Ricordiamo comunque che per SubNet si può anche intendere un troncone di una LAN di grosse dimensioni; idealmente una SubNet è una rete delimitata in cui si fa uso di un protocollo di trasmissione dati su livello fisico di tipo broadcast, situazione in cui è massimo il profitto che si può trarre da un probe di rete.
La comunicazione fisica si realizza mediante i consueti protocolli (IP nel nostro caso); in ogni singola sottorete si parla un qualsiasi protocollo broadcast (Ethernet come Token Ring). A livello logico, i correlatori comunicano tra loro, e solo il correlatore globale parla con la console di management. Per una maggiore affidabilità e semplicità di realizzazione, è opportuno che i correlatori si parlino con un protocollo connection/oriented e affidabile (es. TCP/IP).

ii. Funzionamento

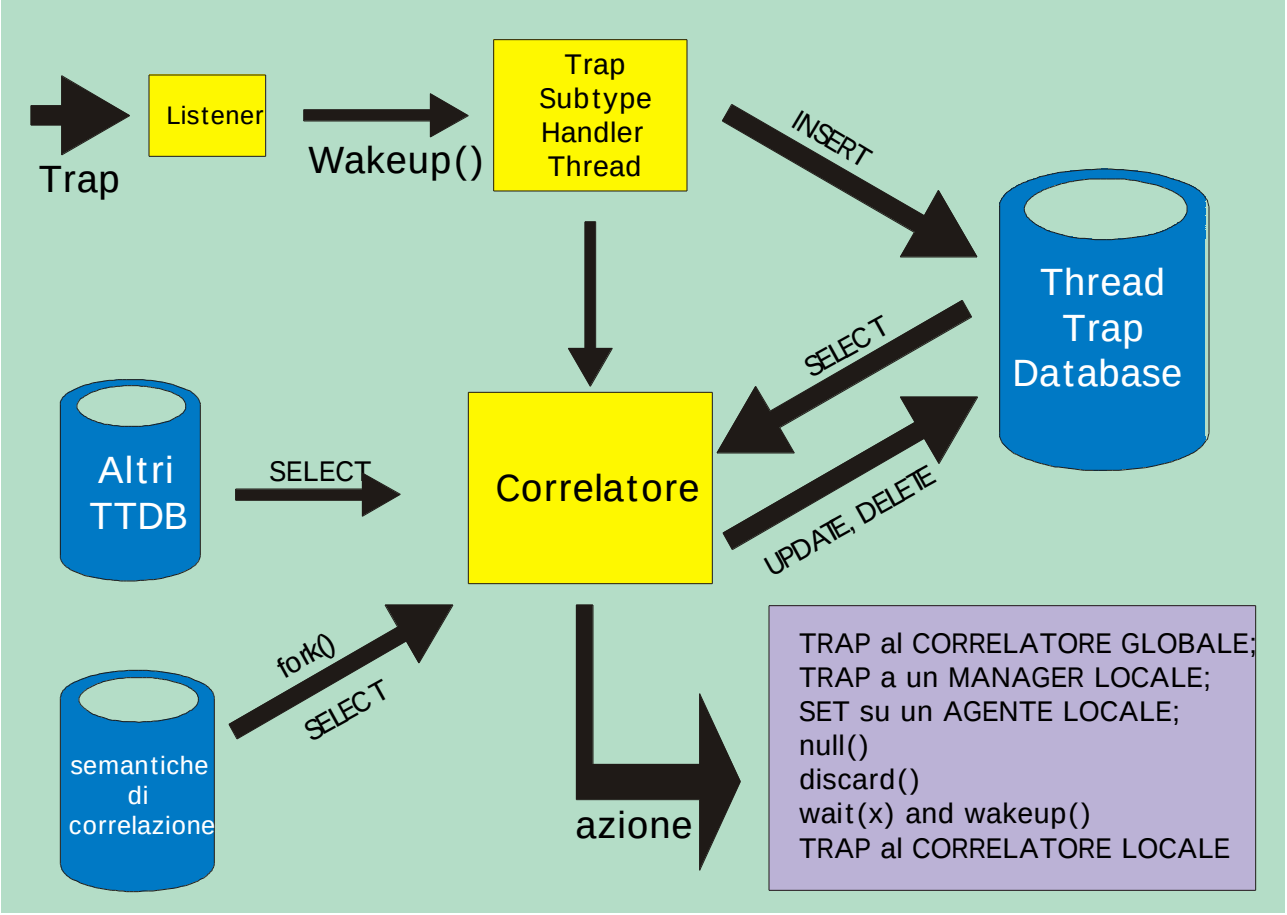
Il sistema si compone di due tipologie di applicativi: un “correlatore globale”, la cui posizione logica è immediatamente “a valle” della console di management, e un “correlatore locale”, la cui posizione logica è all’interno di una unità da gestire, sia essa la stessa rete, un suo troncone o una sua sub-unità, o una zona particolarmente critica per funzionalità o volume di traffico (Es. un cluster di macchine dedicate a una singola funzione).
Il filtraggio delle trap e degli allarmi avviene in modo gerarchico, su due livelli:

1. Locale: il correlatore locale si occupa di ricevere tutti gli allarmi (SNMPv1-trap/SNMPv2c-inform nelle nostre ipotesi) generati da qualsivoglia applicativo in grado di generarle; riceve quindi trap da manager, da agent, e da oggetti “stupidi” come i probes RMON-1;
2. Globale: il correlatore globale si occupa di ricevere informazioni già filtrate dal livello gerarchico inferiore, correlandole tra loro e informando – eventualmente – la console di management di eventuali condizioni che descrivano anomalie critiche.

Entrambi i livelli gerarchici sono anche in grado di svolgere operazioni automatiche (nel nostro caso, delle SNMP-set) per la risoluzione automatica del problema. Si configura in questo caso una sorta di “super-manager” dotato di una intelligenza che lo renda in grado di risolvere i problemi in modo autonomo con minimo intervento umano o di allertare il network manager con dati precisi.

b. Il Correlatore Locale

i. Layout Logico



ii. Funzionamento

La componente Listener resta in attesa per ogni genere di trap: da quelle generate dal probe R-MON (che saranno prevedibilmente le più bisognose di trattamento) fino a quelle generate da altri manager o agent SNMP. È sufficiente introdurre, nella lista di indirizzi delle singole emettenti di trap, anche il socket sul quale sta in ascolto la componente Listener.
La componente Listener determina il tipo di trap, e provvede a “svegliare” il thread corrispondente al tipo di trap, chiamato Trap Subtype Handler Thread (abbreviato in TSHT).
Il TSHT come prima cosa inserisce (mediante una semplice INSERT sql-92 nel nostro caso) la trap nel proprio database di trap, chiamato Thread Trap Database (abbreviato in TTDB), fatto questo passa il controllo alla componente cruciale, il correlatore.
La prima cosa che il correlatore esegue è caricare, per il tipo di trap, l’apposita semantica di correlazione. La semantica di correlazione esprime in dettaglio precondizioni e azioni che si possono verificare confrontando le trap tra di loro. Dal punto di vista dell’implementazione, si può intendere la semantica di correlazione sia come un insieme di regole espresse in un meta-linguaggio (che vengono caricate con una semplice SELECT sql e mandate in esecuzione dal correlatore) che come un vero e proprio segmento di codice che il correlatore deve mandare in esecuzione (espresso con la possibilità di fork() nella relazione tra database delle semantiche e modulo correlatore).

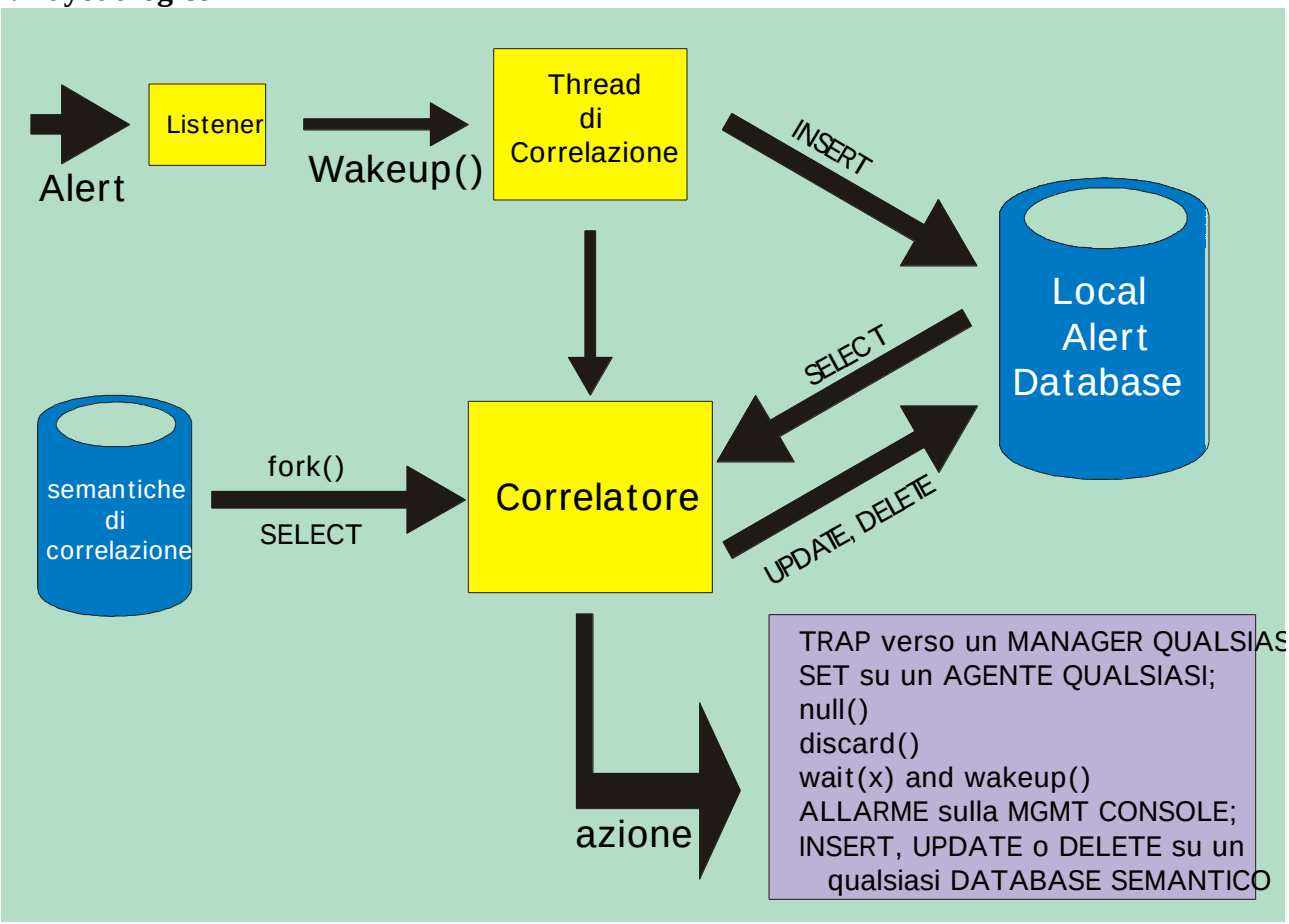
In entrambe le ipotesi, il modulo correlatore può aver bisogno di consultare il database di un altro TSHT (mediante SELECT sql), di consultare il proprio TTDB, o di aggiornarlo. In molti casi si renderanno necessarie tutte queste opzioni. Si possono inoltre caricare ed eseguire più regole semantiche in fase di correlazione.
Alla fine del ciclo di correlazione, il TSHT fornisce un risultato. Detto risultato può essere:

1. ***TRAP al correlatore globale***, nel caso che la correlazione delle trap abbia determinato una condizione anomala che debba essere portata all'attenzione della console di management, o correlata a sua volta con altre condizioni di pari portata. Da notare come (vedi a tal fine la descrizione del correlatore globale) l'uso della trap non è necessario, dal momento che il correlatore globale non presuppone che gli allarmi in ingresso siano SNMPtrap; si può usare qualsiasi tipo di "allarme" la combinazione rete-protocollo ci metta a disposizione: ROSE qualora stiamo operando su uno stack ISO/OSI, o RMI nel caso di un management java-oriented.
2. ***SET su un agente locale***, una o più di una, nel caso in cui il modulo correlatore abbia riconosciuto un *pattern* che può essere risolto in modo automatico senza intervento umano. Questa azione può o non può essere accompagnata da una TRAP al correlatore globale, volta a informare la console di management della problematica riscontrata e della conseguente azione intrapresa in modo automatico. Un tipico esempio di questo comportamento è la realizzazione di una funzionalità di watchdog (ripristino automatico di un servizio dopo che è andato down)
3. ***null()***, quando si decide di tenere la trap ma non fare nulla, in previsione di una futura correlazione con altre trap o, più genericamente, nel caso in cui la trap non meriti una azione immediata ma si possa rendere importante per usi futuri.
4. ***discard()***, quando le semantiche identificano una trap senza senso che deve essere buttata; in questo modo il TSHT può riconoscere anche eventuali forged trap.
5. ***wait(x) and wakeup()***, quando le semantiche identificano che il modulo debba attendere un tempo discrezionale e poi risvegliarsi; questa azione viene intrapresa in casi in cui la trap non ha senso in sé, ma ha senso solo se correlata ad altre trap la cui ricezione è attesa in un intervallo lungo x millisecondi; ad ogni modo, al termine di x millisecondi il correlatore si risveglia e prosegue la sua azione. Questa opzione è utile per una azione appropriata in caso di trap storm.
6. ***TRAP al correlatore locale***, quando la semantica di correlazione mette in luce la necessità di agire in modo attivo su database di altri TSHT; poiché il correlatore può accedere a detti database soltanto in lettura, l'unico modo per operare in scrittura su altri TTDB è di risvegliare il thread che ne è proprietario, opzione attivabile con una trap. Questa opzione è utile qualora si configuri una correlazione molti-a-uno tra trap di due tipi distinti.

I risultati di cui al punto 1-6 non sono mutuamente esclusivi; in alcuni casi si possono intraprendere anche più di un punto, come nel caso delineato al punto 2, o nel caso delineato al punto 4 (informare la console di management nel caso di una forged trap)

c. Il Correlatore Globale

i. Layout logico



ii. Funzionamento

Il correlatore globale, a differenza del correlatore locale, NON riceve allarmi da entità che non siano correlatori locali; anche se il mezzo usato per la comunicazione è una trap SNMP, a livello logico è più appropriato parlare di “alert” che non di trap vere e proprie.
La ricezione di un allarme da parte di un correlatore locale provoca il risveglio del thread dedicato alla correlazione, che in modo identico a un TSHT inserisce l’allarme in un database di allarmi e passa il controllo al correlatore. Da notare che, a differenza del correlatore locale, la minore quantità di allarmi ricevuti, oltre che la assenza di correlatori di “pari livello”, implicano la necessità di un solo thread che gestisca tutti gli allarmi.
Il controllo è sostanzialmente lo stesso di quello effettuato da un correlatore locale; si caricano le regole (se espresse in un metalinguaggio semantico) o i segmenti di codice da eseguire (se nel database sono presenti dei BLOB di codice binario) appropriato al tipo di allarme che si deve gestire. Analogamente al correlatore locale, la fase di correlazione può aver bisogno di consultare, aggiornare o modificare il database di allarmi finora ricevuti dal correlatore globale. La fase di caricamento ed esecuzione di regole semantiche può essere ripetuta più volte. Al termine della fase di valutazione delle semantiche il correlatore globale intraprende una azione. Tra le azioni che il correlatore globale può intraprendere troviamo:

1. ***TRAP verso un QUALSIASI MANAGER***, quando la correlazione degli allarmi identifica una situazione anomala che il correlatore globale può risolvere in modo automatico inviando una o più trap verso manager residenti in uno o più tronconi della rete gestita. Questa azione può essere accompagnata da un avvertimento sulla console di management, che informa il gestore della rete della problematica riscontrata e della azione intrapresa.
2. ***SET verso un QUALSIASI AGENT***, quando la correlazione degli allarmi identifica una situazione anomala che il correlatore globale può risolvere in modo automatico colloquiando direttamente con gli agent scavalcando i manager; questa situazione può verificarsi quando uno o più manager sono down e non è possibile ripristinarli automaticamente. Questa azione può essere accompagnata da un avvertimento sulla console di management, che informa il gestore della rete della problematica riscontrata e della azione intrapresa.
3. ***null()***, quando la ricezione dell'allarme locale e il suo confronto con gli altri non richiedono altro che la sua conservazione nell'apposito database, e nessuna altra azione;
4. ***discard()***, quando l'allarme locale non ha rilevanza (per esempio come conseguenza, dopo un avvenuto *wait(x) and wakeup()*, di una risoluzione della problematica collegata) e non deve essere ignorato.
5. ***ALLARME su MGMT CONSOLE***, quando la correlazione dei vari allarmi ricevuti indica una condizione che il sistema non può risolvere in modo automatico, o un altro errore critico (esempio: disastri) che deve essere risolto in modo manuale dall'operatore. Questa azione può anche essere intrapresa all'atto della risoluzione automatica di un problema segnalato all'operatore ma risolto in modo automatico, per informare l'operatore dell'avvenuta risoluzione della condizione anomale e quindi del rientro dell'allarme ad essa collegata.
6. ***INSERT, UPDATE o DELETE su un qualsiasi DATABASE SEMANTICO***; questa azione permette di configurare il sistema di correlazione globale/locale come un sistema esperto, in grado di aggiornare le metriche in conseguenza di cambiamenti nella rete gestita. Tra le opzioni che possono determinare una simile scelta:
    a. Modifica della topologia della rete che la correlazione precedente ha già riconosciuto come definitiva (es. aggiunta di una nuova macchina, o di una scheda di rete, o di una nuova route verso l'esterno, o una riorganizzazione della rete).
    b. Modifica del traffico della rete, per quantità o tipologia, che la correlazione ha precedentemente riconosciuto come trend confermato nel tempo. Questa condizione può essere collegata alla precedente (es. aumento del traffico http dopo l'aggiunta di due nuove stazioni di lavoro) o può essere indipendente (es. crescita del lavoro di un server Apache corrispondente alla aumentata popolarità dei siti da esso gestiti).
    c. Riconoscimento, nel lungo periodo, di rapporti causa-effetto tra allarme e allarme, o tra allarme e condizioni particolari (es. aumento del traffico sotto Natale, diminuzione in Agosto in corrispondenza delle ferie)

4. Esempi di uso

L'architettura sopra descritta è uno schema di comportamento molto simile a quello del cervello umano, e rappresenta in maniera schematica quello che molti *network manager* farebbero se il sistema non fosse attivo; controllerebbero le informazioni, ne verificherebbero il rapporto causa-effetto, e si muoverebbero in modo appropriato per risolvere il problema e far rientrare l'allarme.
Quello che in un gestore di rete è rappresentato dall'esperienza e dalla capacità diventa, nel sistema qui proposto, la ricchezza e completezza dei vari databases semantici (TTDB); quello che in un gestore di rete è intelligenza, senso logico e capacità di apprendimento diventa, nel correlatore locale, la capacità della componente "auto-aggiornante" derivata dal correlatore di creare nuove semantiche corrette a partire da un sottoinsieme di semantiche ritenute corrette e da rapporti causa-effetto precedentemente individuati; in altre parole, ricchezza del database che contiene la semantica globale.
Ne consegue quindi che il vero cuore dell'applicazione non è la struttura che realizza la applicazione stessa, quando la ricchezza conoscitiva del sistema, intesa come potenza espressiva delle semantiche che il sistema conosce o è in grado di generare.

Nella trattazione del "case study" che segue, tuttavia, non presupporremo nessun tipo di realizzazione, poiché una particolare realizzazione del sistema implicherebbe un formato per la semantica, sia essa un metalinguaggio o vero e proprio codice; analizzeremo a parole come il sistema si comporterebbe, caso per caso, dettagliando di volta in volta le ipotesi che si rendono necessarie per non rendere l'analisi troppo discorsiva e eccessivamente ampia. La semantica dei database viene quindi spiegata, prosaicamente parlando, "a parole".

Ci porremo quindi nell'ipotesi di "Trap Storm" analizzeremo una situazione relativamente tipica, quello della tempesta di trap SNMPv1 "cold start", ipotizzando un blackout di un troncone di sottorete e il reboot successivo, al ritorno della corrente, delle macchine.

a. Cold Start: un esempio di trap storm

Supponiamo di trovarci in un troncone di sottorete – esempio un edificio, oppure una sede distaccata di una azienda – e che ad un certo punto venga a mancare la corrente in tutto il piano. Supponiamo (ma è irrilevante) che nessuna macchina gestita nel piano sia dotata di un gruppo di continuità, ad eccezione di quella su cui gira il correlatore locale, che deve essere sempre attivo e pronto alla stregua di una console di management. Si presume altresì che la rotta verso il correlatore globale sia aperta (basta collegare lo stesso UPS anche al router e all'hub della sottorete).

Dopo un arbitrario tempo X, il correlatore incomincia a ricevere parecchie tipologie di trap SNMP:

1. *Inform* di manager che hanno fatto polling sui rispettivi agent e informano che il polling è fallito (ovvio: il computer su cui gira l'agent è spento);
2. *trap* provenienti dal probe RMON, se attivo, che informa il correlatore locale che il traffico si è improvvisamente azzerato;
3. *trap* provenienti da altri correlatori remoti, che a loro volta sono stati avvertiti dai manager nei loro rispettivi tronconi che non riescono a colloquiare con gli agent nel troncone di rete in cui è andata via la luce, e dopo alcuni tentativi hanno deciso di informare il correlatore locale.

Alla ricezione delle prime trap di questo tipo, il correlatore controlla la sua regola di semantica; sa che se un servizio di una determinata macchina è down, può essere per svariati motivi, incluso il fatto che la macchina sia spenta (è il nostro caso) o che siano andati persi svariati datagram UDP per cui il manager non è riuscito a far polling (come se qualcuno avesse acceso un forno a microonde accanto ad un cavo), o altri motivi. In questo caso il correlatore archivia la trap (con una insert), si accorge che ci sono poche altre trap riconducibili al caso "è andata via la corrente", e si limita ad aspettare (procedura di *wait(x) and wakeup()*)
All'atto del wakeup sono nel frattempo arrivate altre trap. Poniamoci nel caso in cui sia arrivata una trap del 1° tipo descritto sopra. La semantica a questo punto comanda di verificare se per caso non sia andata via la corrente; per fare questo esegue una operazione di SELECT sul TTDB locale, e comanda due operazioni di SELECT, una sul TTDB delle trap del 2° tipo, e una sul TTDB del thread che gestisce le trap del 3° tipo. Poiché dal confronto dei risultati si evince che quasi tutte le macchine del troncone sono down, il correlatore conclude che è andata via la luce.
A questo punto il correlatore locale "si ricorda", in tutti e tre i thread che gestiscono le trap di cui sopra, che è andata via la luce. Alla prima trap dopo questa decisione si occupa di automatizzare la gestione della situazione, con alcune operazioni:

1. SET a un MANAGER LOCALE: intima ai propri manager di smetterla di fare polling sulle risorse che non possono rispondere.
2. TRAP al CORRELATORE GLOBALE, in cui il correlatore locale informa il correlatore globale che sottorete da lui gestita si trova al buio. A questo punto il correlatore globale si muove, seguendo questi rami:
    a. TRAP a un CORRELATORE LOCALE: il correlatore globale, preso atto che la corrente manca, informa tutti gli altri correlatori locali della situazione. I correlatori locali possono quindi intimare ai manager sotto di loro di smetterla di fare polling su macchine spente.
    b. Wait(x) and wakeup(): il correlatore globale decide che non è il caso di disturbare l'operatore per un semplice blackout di corrente (che, per la velocità a cui avvengono

le cose, può anche essere momentaneo, nel qual caso le macchine stanno tutte riavviandosi) e si ripromette di controllare la situazione tra dieci minuti.

A questo punto tutte le successive trap (saranno pochissime, dopo l'intervento del correlatore globale) che indicano la mancata risposta delle macchine al polling verranno conservate, senza prendere alcuna azione (ramo *null()*).

Dopo dieci minuti il correlatore globale si risveglia, e vede che il correlatore locale non lo ha ancora informato del fatto che la corrente può essere tornata. A quel punto informa la console di management, con un messaggio sulla falsariga del "Troncone C (rete di Pisa): la corrente è andata via all'ora <timestamp> e non è ancora tornata dopo dieci minuti. Verificherò tra cinque minuti".

Supponiamo che dopo pochi secondi dall'invio del messaggio sulla console di management la corrente ritorni. Le macchine incominciano a fare il riavvio e a tornare su. Ogni macchina che torna su manda una trap di cold start, che viene recepita dal correlatore locale.
Il correlatore sa che la corrente è andata via: sa che alla prima trap indicante un cold start ne seguiranno probabilmente altre. Ragion per cui, le prime trap di cold start vengono trattate come segue:

1. SELECT sul TTDB corrispondente al TSHT del cold start, per rendersi conto che in effetti ci sono state poche cold start rispetto al numero di macchine morte;
2. SET a un MANAGER LOCALE, nel caso in cui le macchine appena tornate su fossero oggetto di un polling da parte di un manager locale: se la macchina è tornata su, il manager può riprenderne in mano la gestione;
3. null(), poiché vengono archiviate.

Non appena il punto 1 di cui sopra indica che sono tornate su un considerevole numero di macchine, il correlatore locale può con ragionevole certezza informare il correlatore globale che la rete è tornata su perché finalmente è ritornata la corrente. Il correlatore globale, visto che ha avvisato la console di management che il troncone intero era down, informa che il troncone ha di nuovo ricevuto la corrente, con un messaggio del tipo "la corrente è ritornata al troncone C (rete di Pisa)". Quando il thread si risveglia al tempo programmato per verificare se la corrente è ritornata, si accorge dell'allarme inviato a suo tempo dal correlatore locale e non duplica quindi l'informazione sulla console di management. Informa inoltre tutti gli altri correlatori locali che il troncone in oggetto è di nuovo su, e che i manager gerarchicamente sotto di loro possono tranquillamente ritornare a fare polling su eventuali risorse ivi situate. Fatto questo, dal punto di vista del correlatore globale la pratica è chiusa.

Il correlatore locale, tuttavia, deve fare ancora dei controlli. Continua a ricevere trap di cold start dalle macchine nella sua sottorete, e ogni volta che ne riceve una controlla che ad ogni macchina dichiarata "morta" dalle trap in precedenza ricevuta corrisponda una trap di cold start; alla ricezione di ogni trap di cold start si limita quindi a fare la SELECT e a fare una *wait(x) and wakeup()*, per verificare la situazione dopo alcuni secondi.

Dopo un tempo arbitrario (corrispondente al tempo di reboot più lungo della macchina più lenta, più un margine di sicurezza) il thread si risveglia e si accorge, mediante debite SELECT, che le macchine si sono risvegliate tutte; tutte tranne una. Comanda ad un manager di fare il reboot manuale della macchina (supponendo che la macchina abbia capacità di wake-on-lan) e si ripromette di fare polling (*wait(x) and wakeup()*) dopo dieci minuti.

Dopo dieci minuti il thread del cold start si risveglia, per trovare nel database delle trap un avviso da parte del manager che non riesce a riavviare la macchina, e nessuna cold start proveniente dalla macchina. A questo punto deve di nuovo informare il correlatore globale che la macchina non vuol saperne di venire su, e che un tentativo automatico di rimetterla su è fallito.
Il correlatore globale riceve l'allarme e controlla, tramite SELECT nel suo database, che il troncone della macchina è andato giù per mancanza di corrente trenta minuti fa, ed è tornato su dopo dodici. La semantica in questo caso prescrive di avvertire l'operatore (dal momento che le procedure automatiche hanno fallito), con un allarme sulla console di management del tipo "la macchina X non è tornata su dopo un black-out. Verificare".
In questo caso il gestore della rete è costretto ad alzare il telefono e a telefonare a Pisa, per scoprire che in effetti la macchina X non può oggettivamente tornare su, in quanto lo sbalzo di corrente ha danneggiato irreparabilmente l'alimentatore.

Il sistema è stato in grado di gestire un numero elevatissimo di trap SNMP provenienti da punti diversi di una rete che può anche essere estesa per tutto il territorio nazionale, gestendo in modo autonomo la situazione, dando soltanto due avvertimenti e un allarme "critical" sulla console di management, e fornendo una diagnostica relativamente precisa del disastro occorso dopo il blackout; la semantica di gestione usa poche regole, ed è stata formulata nel case study per gestire in maniera esatta questa problematica. Una semantica accuratamente espressa può gestire tutti i casi possibili di trap storm con un numero ridotto di regole dalla formulazione relativamente semplice.

Supponiamo un downtime totale di dodici minuti di blackout, più un tempo medio di reboot di tre minuti (tra macchina e eventuali agent ivi presenti). Supponiamo che il troncone sia composto di trenta macchine, che su ogni macchina giri almeno un agent, e che altri dieci manager provenienti da cinque altri diversi tronconi gestiscano risorse site nel troncone C. Supponiamo inoltre un intervallo di polling di 30 secondi. In assenza del sistema di correlazione avrei generato:

30 * 15 = 450 trap provenienti da manager che non riescono più a fare polling, avvertendo la console di management;
10 * 15 = 150 trap provenienti da altri manager, da vari sistemi, che non riescono a fare polling, avvertendo la console di management;
29 trap di cold start riportate sulla console di management;
un paio di trap proveninente dall'agent che gestisce l'UPS al quale sono stati collocati router, hub e probe RMON, in cui si informa la console di management che l'UPS è entrato in funzione;
una trentina di trap da un probe RMON che informa che il traffico della sua rete è sceso a zero.
Altre trap di un manager impossibilitato a parlare con la macchina X, down per guasto all'alimentatore.

Tralasciando altri tipi di trap connessi alla situazione in esame, la console di management avrebbe ricevuto poco meno di 700 allarmi, contro i 3 ricevuti nel caso di presenza di correlatore. In un caso del genere il sistema è in grado di ridurre gli allarmi all'incirca di un fattore 233.

5. Sviluppi e limiti dell'architettura

L'architettura proposta rappresenta una istantanea "a blocchi" del metodo di ragionamento umano. Presenta quindi gli stessi limiti dell'apprendimento e dell'esperienza umana. In particolar modo:

1. Non si può saper tutto;
2. Non si può imparare tutto;
3. Non sempre la soluzione è lampante e immediata.

In questa ipotesi le problematiche dell'architettura diventano immediatamente problematiche di semantica. Il punto di forza del sistema sta nella semantica; il punto debole pure.

Una semantica "generale" permette di avere un sistema veloce e snello, ma non permette di raggiungere prestazioni ottime come quelle delineate nel case study; una semantica "esatta" permette di diagnosticare molte situazioni in tempo quasi zero, semplificare notevolmente il lavoro del network manager, ma nel contempo crea un sistema più pesante, che si trova ad eseguire molto lavoro ad ogni singola trap, poiché deve verificare molte regole alla volta.

L'approccio più conveniente è quindi un approccio misto, con l'applicazione di alcune regole semantiche molto "generalizzate", che consentano una riduzione considerevole della maggior parte dei casi senza generare troppo overhead, riservando una semantica più esatta per servizi o situazioni che per frequenza o criticità devono essere gestite nel migliore dei modi possibili.

Da questo punto di vista una delle chiavi di volta è la opzione, fornita dal correlatore globale, che permette la trasformazione del sistema, che può essere visto come una macchina di Turing aventi come istruzioni le trap, a un sistema in grado di apprendere e autoprogrammarsi, partendo da un insieme di istruzioni leggero e veloce per arrivare ad auto-modellarsi sulla realtà specifica della rete da gestire con l'esperienza. La trattazione delle tecniche necessarie a raggiungere questo risultato esula dal contesto di questo documento; tuttavia questa opzione è la vera e propria chiave di volta per il successo di un sistema di questo genere.

È inoltre importante che le componenti del sistema siano realizzate in modo da essere auto-adattative e il più possibile platform-independant. Alla luce di questa considerazione, e da un punto di vista tecnologico, una evoluzione di questo progetto potrebbe essere una prototipazione dei correlatori locali e globali in linguaggi come C/C++ e Java. Ogni linguaggio ha i suoi pro e i suoi contro:

> C/C++ permette di ottenere il massimo livello di performance dalle macchine, oltre a rendere il caricamento delle regole semantiche – viste come spezzoni di codice – molto più semplice, trattandosi di un meccanismo naturale nel linguaggio (è esattamente il funzionamento della fork: caricare un pezzo di codice oggetto e eseguirlo). Tra i difetti di una prototipazione in C/C++, troviamo il maggior tempo di scrittura di programma e semantiche, che si innalza ulteriormente se per esigenze di portabilità si sceglie di lavorare in ANSI C puro. Inoltre, la necessità di una compilazione "in loco" del codice sulla macchina sulla quale detto codice deve girare mi preclude la possibilità di "distribuire" il carico applicativo su diverse macchine, come potrei fare invece se avessi a disposizione il meccanismo di invocazione remota di Java.

Java ha tra i suoi pregi un tempo di sviluppo molto inferiore, un paradigma di programmazione OOP che permette di generare codice più compatto, mantenibile e più rispondente nella sua stessa natura alla idea che sta alla base dell'applicazione; inoltre si ha la possibilità di appoggiarsi alle API JMX per realizzare – internamente – operazioni di management, e un ottimo supporto all'invocazione di metodi remoti, con RMI (che semplifica la realizzazione del correlatore globale). Java inoltre è un linguaggio che strizza un occhio alla sicurezza. Tuttavia, la performance di Java è quantomeno risibile, e la necessità di caricare le regole semantiche in forma testuale e interpretarle di volta in volta innalzano particolarmente la complessità del progetto (la complessità di un interprete di questo tipo è paragonabile infatti a quella di un compilatore).

Nonostante la complessità del progetto, ritengo che l'architettura descritta sia sostanzialmente in grado di semplificare notevolmente la vita nella stragrande maggioranza delle problematiche di network management; il livello di performance raggiungibile, sia in termini di efficacia che in termini di efficienza, dipende unicamente dal bagaglio culturale che è trasmesso all'applicazione sotto forma di regole semantiche. Se la realizzazione permette di superare il naturale ostacolo del resource hogging associabile ad una applicazione così complessa, gli sviluppi futuri possono essere importanti.

6. Riferimenti

RFC rilevanti relativi al case study e alle supposizioni:

1. 1757, RMON-1 MIB;
2. 2021, RMON-2 MIB;
3. 1157, SNMPv1;
4. 1905, SNMPv2c;

reperibili on line su www.ietf.org (sito Internet Engineering Task Force)

Articoli di interesse relativi al case study, o possibili espansioni/integrazioni del sistema:

Mark Sylor, Lingmin Meng, "Using Time Over Threshold to Reduce Noise in Performance and Fault Management Systems", Boston University, 2001: una trattazione del problema delle metriche e della generazione di allarmi relativi al superamento dei threshold, con una spiegazione degli algoritmi volti a ridurre il numero di trap;
Ricaldo Vilalta, Sheng Ma, Joseph Hellerstein, "Rule Induction of Computer Events", IBM T.J. Watson Research Centre, 2001: un articolo interessante sulla deduzione di regole euristiche (non esatte) a partire da una determinata classe di eventi, raffrontata con eventi precedenti in una determinata time window.
Mark Brode, Irina Rish, Sheng Ma, "Optimizing Probe Selection for Fault Localization", IBM T.J. Watson Research Centre, 2001: un articolo che descrive alcuni algoritmi e pattern per il piazzamento strategico di probe distribuiti, volto a raggiungere il massimo grado di efficienza con il minor numero di probe possibili (e quindi minor traffico da essi generato). Applicabile al piazzamento degli eventuali correlatori locali in un sistema che risponda alla architettura qui descritta.