Netgraph

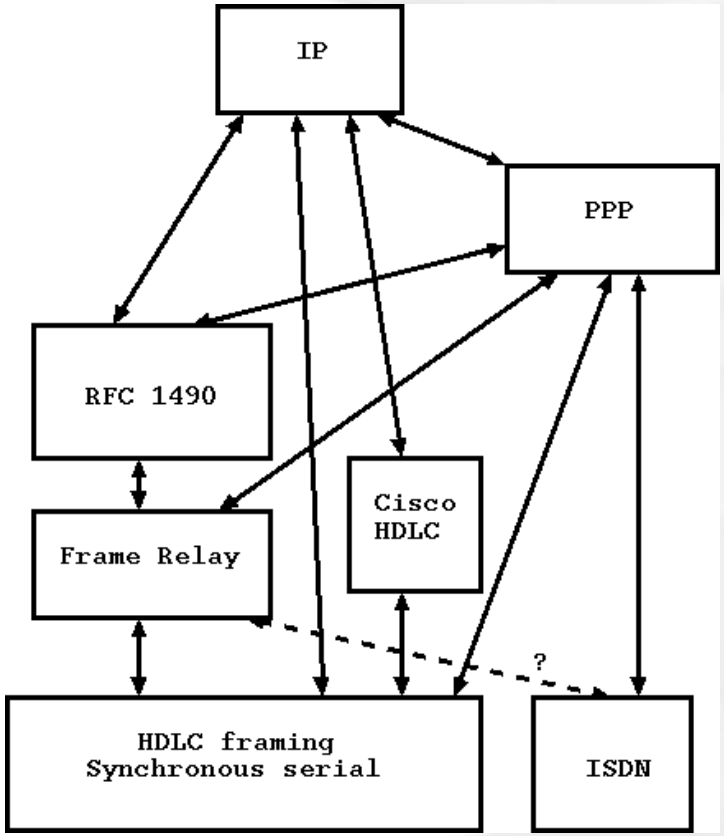
Presentazione del sistema ed esempi di utilizzo

2 ottobre 2004, GUFICon #5

Dario Freni <saturnero@gufi.org>

G.U.F.I. - Gruppo Utenti FreeBSD Italia

Netgraph



Ideato nel 1996 da
Julian Elischer <julian@FreeBSD.org> e
Archie Cobbs <archie@FreeBSD.org>

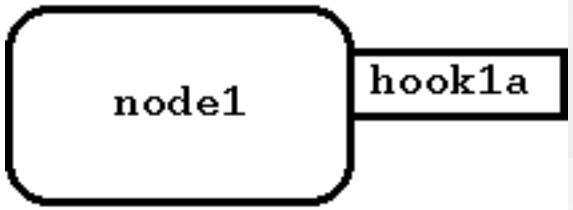
Framework uniforme e modulare per l'implementazione di oggetti (nodi), in ambiente kernel.

Studiato per semplificare lo sviluppo del supporto a protocolli diversi, nonché per renderne efficiente la coesistenza.

Attuali esempi di utilizzo: PPPoE, Bluetooth;

È veloce, versatile ed affidabile.

Nodi e hook



L'elemento fondamentale di Netgraph è il *nodo*.

Il nodo è un oggetto, gira in ambiente kernel, è pensato per comunicare con altri nodi.

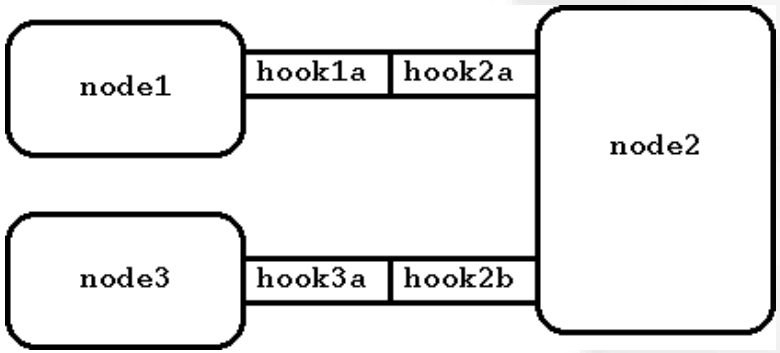
Alcuni nodi sono **semi-permanenti**, ovvero esistono fino a quando non sono più connessi ad altri nodi.

Altri sono **persistenti**, tipicamente associati con qualche device hardware (es.: scheda di rete, dongle Bluetooth).

Ogni nodo fornisce almeno un *hook*, per stabilire canali di comunicazione con altri nodi.

Nodi e hook

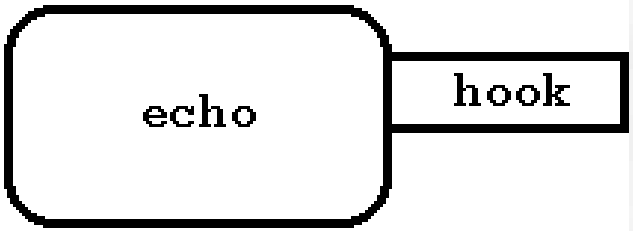
Ogni collegamento fra nodi avviene mediante coppie di hook.



Un nodo fornisce uno o più hook. Gli hook sono identificati univocamente dai loro nomi.

I dati attraversano gli hook sotto forma di pacchetti, strutturati secondo mbuf(9).

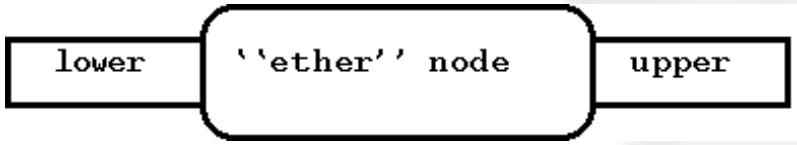
ng_echo



Nodo esemplificativo. Può avere un numero arbitrario di hook.

Alla ricezione di un pacchetto, si limita alla restituzione del pacchetto stesso sull'hook di provenienza.

ng_ether



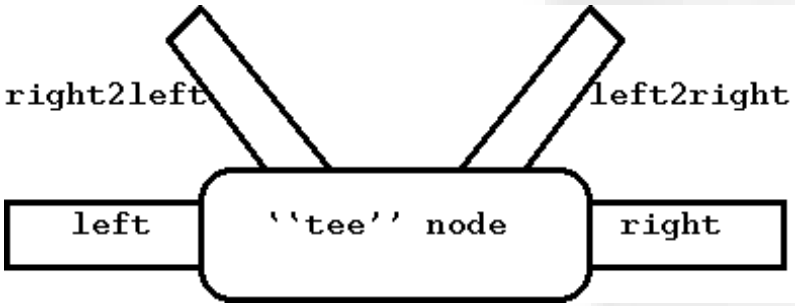
È un nodo permanente, rappresenta una scheda di rete fisica all'interno del sistema Netgraph.

Una volta caricato l'opportuno modulo del kernel, per ogni scheda di rete esiste un nodo associato. Ogni nodo assume lo stesso nome della scheda di rete.

Fornisce due hook, lower e upper.

Il traffico esterno, in entrata, passa da lower.
Il traffico proveniente dal sistema operativo, passa da upper.
I pacchetti immessi in lower finiscono all'esterno.
I pacchetti immessi in upper finiscono al S.O.

ng_tee



Ispirato al comando tee(1) di UNIX.

I pacchetti passano da left a right e viceversa, e vengono duplicati su left2right o right2left a seconda della direzione.

Strumenti di controllo

ngctl – istanzia e controlla i nodi da userland.

Ha un prompt dei comandi, può essere facilmente integrato in script di shell. Sintassi un po' ostica.

nghook – aggancia un hook e stampa l'output a video.

Utile per gestire il traffico da userland.

Comodo soprattutto per debug!

Esempi (1)

Esempi pratici:

```
kldload netgraph        # Carico il framework
kldload ng_ether        # Carico i moduli relativi ai nodi (automatico in 5.x)
kldload ng_tee

nghook -a wi0: lower  # Primo esempio


ngctl mkpeer wi0: tee lower right
ngctl name wi0:lower mytee              # per comodità
ngctl connect wi0: mytee: upper left;

nghook -a tee: right2left
```

Comunicazione con utente

Supporto per i “messaggi” ai nodi (ngctl msg)

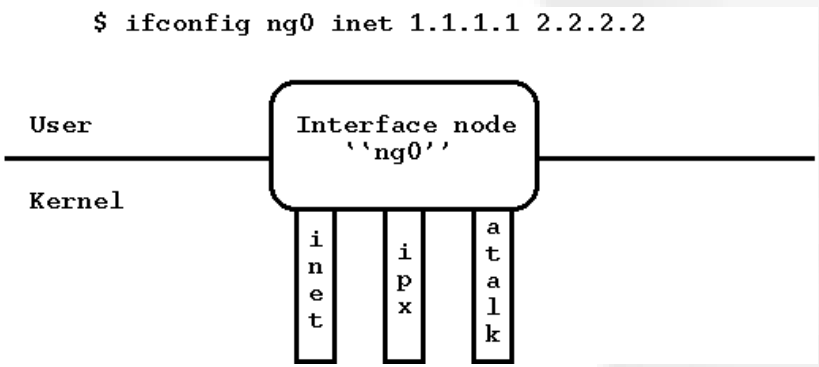
Utilizzato quasi sempre per statistiche, spesso per configurazioni a run-time.

Codifica degli argomenti dei messaggi secondo una sintassi intuitiva.
(per i curiosi: commenti in /usr/include/netgraph/ng_parse.h)

Manca uno standard per sapere a priori quali messaggi sono supportati da un nodo, e la relativa sintassi di chiamata.

Quasi sempre presenti: getstats, clrstats, getclrstats.

ng_iface



Crea una interfaccia di rete artificiale (nomenclata ng0, ng1, ...).

Ogni hook rappresenta un protocollo diverso.

Attualmente presenti: inet, inet6, atalk, ipx, ...

Una volta istanziato si configura come una normale interfaccia di rete point-to-point.

Per simulare interfacce ethernet esiste ng_eiface.

ng_ksocket

Stabilisce una connessione socket (a livello kernel!)

Ha un singolo hook, attraverso il quale passa il traffico in ingresso e in uscita della suddetta socket.

Il nome dell'hook è nel formato <family>/<type>/<proto>, i tre parametri sono gli stessi della syscall socket(2). Es: inet/dgram/udp

Una volta istanziato e agganciato si configura a run-time attraverso messaggi.
Fondamentali:

bind inet/${PROPRIOIP}:${PORTA}
connect inet/${IPREMOTO}:${PORTA}

Esempi(2)

(da /usr/share/examples/netgraph/udp.tunnel)

ngctl mkpeer iface dummy inet

ngctl msg ng0:inet bind inet/:${UDP_TUNNEL_PORT}

ngctl msg ng0:inet connect inet/${REM_EXTERIOR_IP}:
${UDP_TUNNEL_PORT}

ifconfig ng0 ${LOC_INTERIOR_IP} ${REM_INTERIOR_IP}

(stessa configurazione su due macchine... cosa ho fatto?)

XNF

XNF (eXtensible Netgraph Framework)

Infrastruttura modulare per il controllo e la manipolazione del traffico di rete.

È pensato per lo sviluppo di nodi semplici, segmentando il lavoro in più moduli.

Si aggancia ad un nodo ng_ether, e predispone l'utilizzo di due maglie di nodi, una per l'input ed una per l'output.

Ogni nuovo nodo soddisferà determinati prerequisiti, per fornire al sistema le sue opzioni di configurazione.

XNF(2)

La disposizione dei nodi sarà quindi rappresentabile mediante un grafo.

Disposizione dei nodi, e configurazione di ogni singolo nodo, si possono scrivere in un file di configurazione in formato XML, per favorire lo sviluppo di applicazioni che generino una configurazione.

XNF è principalmente pensato come piattaforma per ricerca e sviluppo, semplificando il lavoro del programmatore che vuole sperimentare in ambiente di rete ad un livello basso.

Bibliografia

A. Cobbs, *All About Netgraph*:
http://www.daemonnews.org/200003/netgraph.html

man 4 netgraph

XNF: http://netdev.usr.dico.unimi.it/vpages/XNF

Grazie ;-)