

*Questa è la DEDICA:
ognuno può scrivere quello che vuole,
anche nulla . . .*

Introduzione

Nell'ambito delle reti negli ultimi anni con l'avvento di internet si è assistito ad una crescita esponenziale delle apparecchiature connesse a questa rete. Fino ad oggi questa immensa rete è stata gestita per lo più attraverso indirizzamento delle interfacce di rete di cui i computer sono dotati.

Al momento però ci si trova a fare i conti con la carenza degli indirizzi e pian piano si procede verso una migrazione basata sul nuovo (ma non troppo) protocollo IPv6 che prevede uno spazio di indirizzamento molto più ampio che lascia spazio a nuove idee impraticabili con protocollo IPv4. In seguito sentiremo parlare di IoT (Internet of Things) ed IoTh (Internet of Threads), due paradigmi che possiamo affermare essere il risultato di un'evoluzione sempre crescente dei sistemi informativi.

L'elaborato sarà strutturato in capitoli.

Il primo di questi si preme di introdurre il lettore allo stato attuale dei protocolli di rete e del paradigma IoTh; nel secondo capitolo verrà trattato il progetto VsnLib, ovvero la libreria oggetto della tesi; in fine il terzo capitolo si occuperà di descrivere alcuni esempi di utilizzo della stessa con immagini dimostrative ed istruzioni.

Indice

Introduzione	i
1 Internet of Threads	1
1.1 Paradigma	1
1.2 Stack Implementati	3
1.2.1 PicoTCP	3
1.2.2 LWIP	3
1.2.3 LWIPv6	4
1.3 Netlink	5
1.3.1 Inter Process Communication	5
1.3.2 Libnl	6
2 VsnLib	9
2.1 Il Progetto	9
2.2 VnsLib	10
2.2.1 Preambolo	10
2.2.2 Ambiente	10
2.2.3 Core	11
2.2.4 Moduli	11

2.3	Sviluppi Futuri	11
3	Casi d'uso	13
3.1	Esempi	13
	Conclusioni	15
A	Prima Appendice	17
B	Seconda Appendice	19
	Bibliografia	21

Elenco delle figure

1.1	physical interface stack	2
1.2	virtual stack	2
1.3	mappa concettuale libreria	6
2.1	mappa concettuale libreria	12

Elenco delle tabelle

Capitolo 1

Internet of Threads

1.1 Paradigma

IoT (Internet of Things), ovvero l'internet delle cose.

Questo concetto vuole rappresntare la diffusione dei sistemi embedded come veri e propri nodi di rete, possiamo definirlo il precursore del sistema che stiamo per descrivere ed è il sistema che ad oggi ha permesso di incontrare il nostro condizionatore o la nostra caldaia, ma addirittura il nostro tostapane, in rete.

Ad un certo punto si è sentita l'esigenza di elevare questa astrazione, nasce il concetto di IoTh.

L'idea diventa quella di avere processi come nodi di rete e non oggetti fisici, l'analogia è molto simile alla differenza tra telefoni fissi e telefoni cellulari, ovvero in passata era necessario pensare al luogo in cui una persona potesse trovarsi,, mentre attraverso l'assegnamento di un numero personale oggi possiamo comunicare direttamente con la persona desiderata[1, 2].

In termini di internet questo si traduce nella possibilità di migrare servizi

da un capo all'altro del mondo con la semplicità di un kill and start. Come possiamo notare infatti, con questa tecnologia lo stack " " direttamente integrato nel programma.

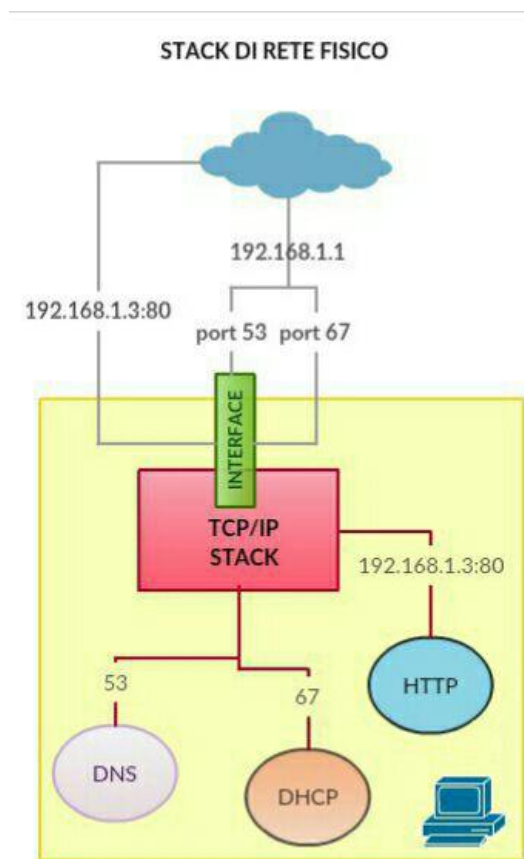


Figura 1.1: stack associato all'interfaccia fisica

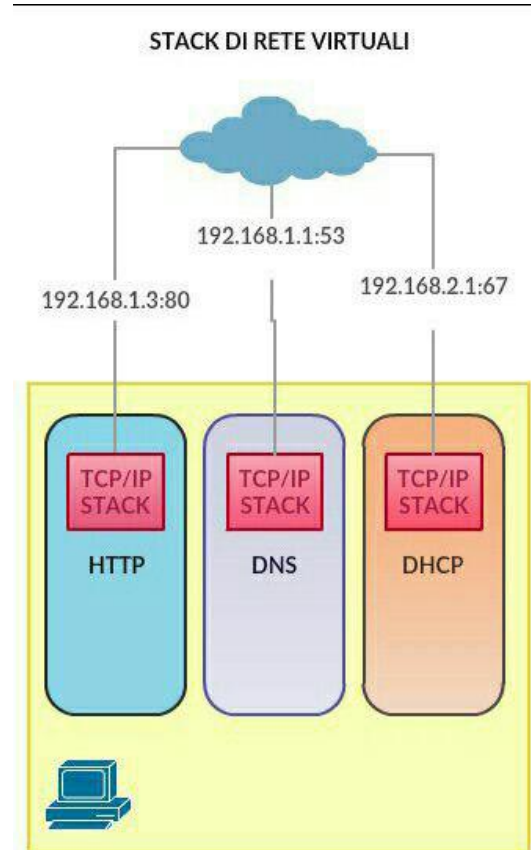


Figura 1.2: stack associato ai processi

Una nota va anche fatta in termini di sicurezza, ogni servizio può essere eseguito con il proprio stack di rete e questo impedirebbe a software di port mapping di carpire informazioni sulla macchina che ospita detto servizio. Inoltre possedendo il proprio stack di rete possiamo eseguire i processi

con un utente non privilegiato e pertanto anche un bug del demone non comprometterebbe l'intero sistema.

1.2 Stack Implementati

Diversi sono i progetti che si occupano di offrire ai processi il proprio stack di rete; tra questi ne verranno presi in esame tre, considerati più indicati per uno studio in quanto open source, Ognuno di essi offre la propria implementazione di stack di rete.

Uno stack di rete legato al processo permette di connettere lo stesso ad una rete reale, tramite le interfacce fornite dal sistema, o virtuale (ad esempio VDE) come se fosse una macchina fisica a se stante, a questo punto ogni processo può staccarsi dal modo in cui la macchina che lo ospita gestisce la rete ed avere le proprie regole.

1.2.1 PicoTCP

Supportato da Tass Belgium (Altran); è lo stack più conosciuto e diffuso per sistemi embedded ed esistono anche dei progetti basati su IP di reti mesh realizzati con picoTCP¹.

1.2.2 LWIP

Nasce dal progetto di Adam Dunkels pensato per sistemi embedded ed inizialmente non forniva supporto ad IPv6.

¹<http://www.picotcp.com/mesh-design-guide>

Light-weight IP vanta un core molto piccolo e la possibilità di eseguire anche senza sistema operativo (single-threaded) e con il minor consumo di RAM possibile, inoltre è modulare ed offre la possibilità di aggiungere protocolli come DHCP e DNS solo in caso di necessità.

Supporta sia little che big endian e gira su processori a 8 e 32 bit, quindi funziona anche su un semplice ATMEGA.

Unica pecca è il protocollo IPv6 che è ancora in fase sperimentale, infatti il supporto IPv6 può essere aggiunto ed è scaricabile da git.

1.2.3 LWIPv6

Precedemente abbiamo fatto notare che LWIP non aveva supporto IPv6 e che tutt'ora è ancora in fase sperimentale, ed Ã" questo uno dei motivi della nascita di LWIPv6.

Quando al team di virtual square è sopravvenuta la necessità di avere uno stack per la rete vde non esisteva ancora nulla di maturo o comunque plug and play, da questa esigenza il team ha pensato di realizzare uno stack versatile in versione libreria tale per cui ogni processo potesse avere il proprio stack di rete.

Caratteristica principale di LWIPv6 è il suo motore ibrido che, di fatto funziona solo con IPv6 e mappa in questo protocollo le comunicazioni IPv4 utilizzando i primi 80 bit dell'indirizzo come flag di riconoscimento con tutti e 80 impostati a 1, ad esempio se volessimo rappresentare la maschera 255.255.255.0, che in esadecimale è equivalente a 0xffff00 e la sua rappresentazione in IPv6 secondo LWIPv6 sarà la seguente 0xffffff.ffffff.ffffff.fffFFFFF.ffff00.

1.3 Netlink

Netlink è un sistema IPC (Inter Process Communication) usato perchè in grado di mettere in comunicazione diversi task, solitamente serve per far comunicare task in user-space con task in kernel-space ma può far interagire anche processi entrambi in user-space.

Studiato per essere il successore di `ioctl` per le configurazioni ed il monitoraggio, si propone di essere più flessibile. Ma come avviene esattamente la comunicazione attraverso netlink?

Netlink utilizza un sistema di socket indicizzato in base ai processi ogni processo può definire socket diversi di tipo netlink (`AF_NETLINK`, `NETLINK_GENERIC`, `NETLINK_XFRM`) per inviare e ricevere messaggi con e da altri processi, implementa un sistema di porte basato sul process ID dei processi; ad esempio per la comunicazione con il kernel viene usata la porta 0.

L'immagine successiva rende perfettamente il concetto, ed è una rielaborazione (per puri motivi stilistici) dell'immagine della documentazione di `libnl`. (reperibile qui²) L'interfaccia di comunicazione è abbastanza standardizzata ma personalizzabile a livello di payload, ognuno può definire una struttura personalizzata per poi usarla per comunicare attraverso i socket netlink con altri processi.

1.3.1 Inter Process Communication

Molti processi necessitano di scambiare informazioni per i motivi più disparati, dalla comunicazione e di rete a quella interna ad una sola mac-

²<https://www.infradead.org/tgr/libnl/doc/images//addressing.png>

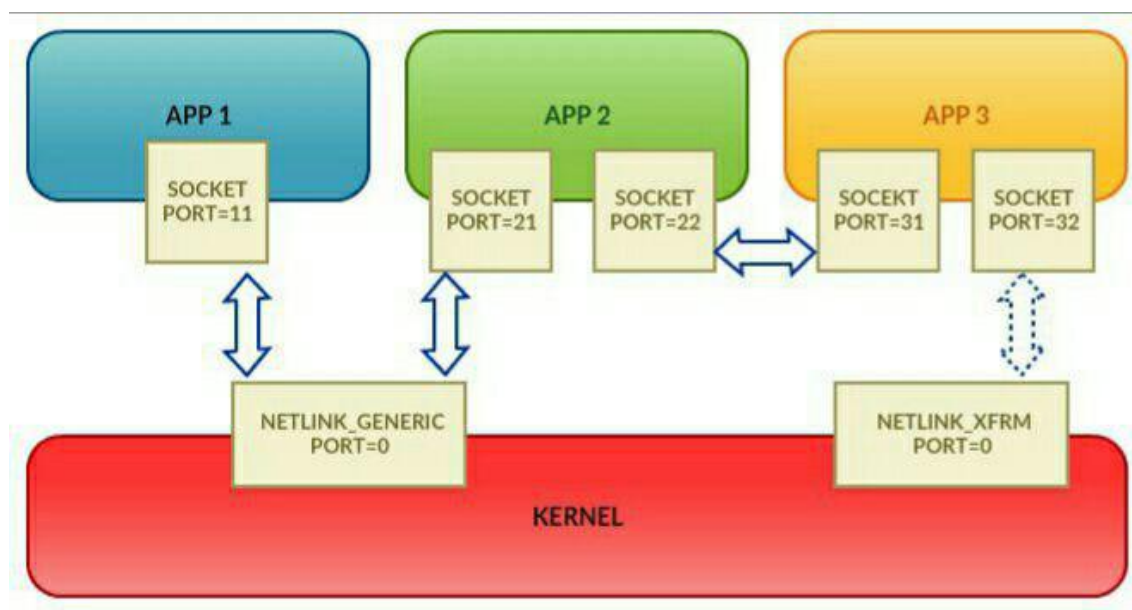


Figura 1.3: comunicazione netlink

china.

Viene da se pensare che costruire programmi che non interagiscono con il mondo esterno o con altri programmi sarebbe una risorsa limitata.

Con IPC intendiamo quindi l'insieme delle tecnologie adottate per permettere ai processi di comunicare tra essi, che siano ospitati sulla stessa macchina o distribuiti sulla rete, tra queste tecnologie esiste appunto quella utilizzata all'interno della libreria oggetto di questo elaborato.

1.3.2 Libnl

Netlink Protocol Library Suite (libnl), è una raccolta di librerie ed utility che forniscono API di comunicazione netlink basate su quelle del kernel linux e comprende [10]:

libnl Core della libreria, offre uno strato di unificazione delle interfacce sulle quali poi si basano le altre librerie che pertanto dipendono da questa;

libnl-route Questa libreria si occupa di fornire API per la configurazione degli elementi della famiglia NETLINK_ROUTE;

libnl-genl genl significa generic netlink e questa libreria offre una versione estesa del protocollo netlink;

libnl-nf API per configurazioni netlink basate su netfilter.

Capitolo 2

VsnLib

Strato di compatibilità tra pacchetti netlink e stack di rete virtuali.

2.1 Il Progetto

Gli stack analizzati in precedenza offrono a grandi linee la stessa tipologia di servizio seppur ognuna con le proprie caratteristiche.

Nessuno dei progetti ha però tenuto in considerazione l'idea di utilizzare un'interfaccia di configurazione che fosse standard e pertanto è necessario usare le funzioni specifiche per ognuno di questi progetti, costringendo il programmatore a cambiare modalità di interazione da stack a stack.

Il progetto di creare questa libreria nasce proprio da questa esigenza, ovvero cercare di uniformare le interfacce di comunicazione in modo tale che l'utente non sia costretto ad adattarsi ogni qual volta decida di cambiare stack.

2.2 VnsLib

2.2.1 Preambolo

Per configurare uno stack di rete programmi come ip generano un pacchetto netlink con le informazioni necessarie lo inviano al kernel che lo elabora esegue le operazioni e genera un messaggio di errore nel quale esiste un flag contenente il numero di errore generato (0 in caso di successo), a questo messaggio è associato un payload di risposta che può essere anche un feedback per l'utente che sta interagendo con il programma.

2.2.2 Ambiente

La necessità primaria era quella di catturare questi netlink e per farlo la strada era quella di intercettare la system call di invio della richiesta contenente appunto il payload, questo momento è quello in cui viene eseguita la sendto.

In questo punto interviene la nostra libreria, che si interpone e di fatto fa le veci del kernel, al quale il pacchetto netlink non arriverà mai realmente. Il progetto inizialmente utilizzava come motore di cattura delle system call la libreria purelibc¹, in seguito però si è giunti alla conclusione che costruire un modulo ad hoc all'interno del sistema vuos fosse un'alternativa più versatile, immediata e pulita, inoltre vuos ha un sistema di debug built in che rende più semplice l'individuazione dei problemi legati allo sviluppo.

Il modulo in questione si chiama unrealvsnlib per analogia alla libreria ma

¹<http://wiki.virtualsquare.org/wiki/index.php/PureLibc>

ognuno potrebbe costruire un suo modulo a seconda di quello che intende controllare.

2.2.3 Core

La libreria risulta come uno strato di compatibilità tra netlink e le specifiche configurazioni degli stack.

Vediamo come è composta:

VnsLib: Il primo stato si occupa di inizializzare la libreria in base allo stack che si intende utilizzare, in questo modo possiamo caricare dinamicamente solo il modulo che ci interessa.

Modules: Sono la parte specifica della libreria, essi contengono l'intestazione delle funzioni generiche che si occupano di chiamare quelle particolari per ogni stack.

2.2.4 Moduli

La potenza dei moduli risiede nel fatto che chiunque abbia intenzione di costruire uno stack personalizzato, o di utilizzarne uno per il quale non esiste un modulo, può facilmente realizzare il proprio e la libreria si occuperà di farne il caricamento qualora fosse richiesto.

2.3 Sviluppi Futuri

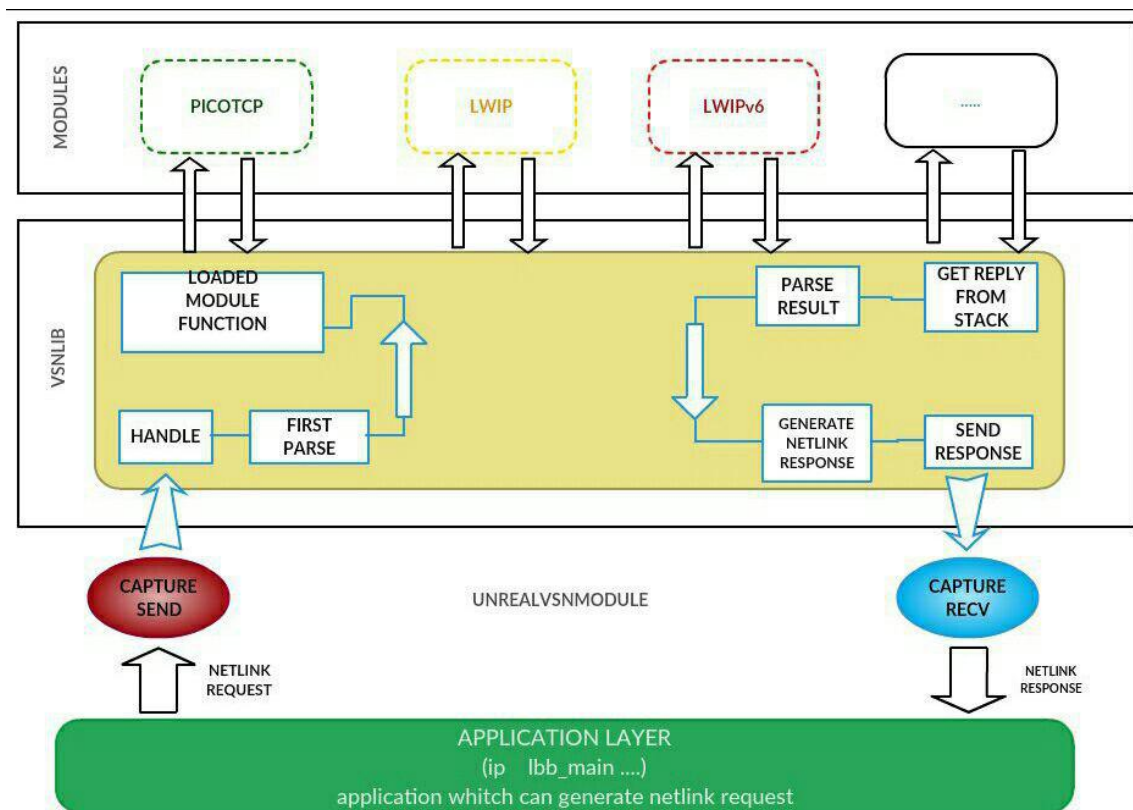


Figura 2.1: mappa concettuale libreria

Capitolo 3

Casi d'uso

Di seguito alcuni esempi di utilizzo della libreria. Le dimostrazioni seguenti sono state effettuate su una macchina con architettura amd64 e con installato il sistema operativo debian in versione sid (quindi unstable) ma sono state testate e riprodotte anche su altre configurazioni e distribuzioni GNU/Linux.

3.1 Esempi

Conclusioni

L'utilizzo di queste tipologie di stack dipende strettamente dalla diffusione del protocollo IPv6 e dei sistemi embedded e sembra essere la direzione in cui questo settore sta muovendosi. La ricerca e lo sviluppo di questi meccanismi sono necessari ad ottenere un sistema performante e stabile quando IPv6 sarà l'ordinario, vanno poi considerati anche i vantaggi conseguenti nelle reti virtuali che già ora vengono utilizzate per la sperimentazione.

L'IoTh ne è un'astrazione più forte. In queste conclusioni voglio fare un riferimento alla bibliografia: questo è il mio riferimento [3, 4].

Appendice A

Prima Appendice

In questa Appendice non si è utilizzato il comando:
`\clearpage{\pagestyle{empty}\cleardoublepage}`, ed infatti l'ultima pagina 8 ha l'intestazione con il numero di pagina in alto.

Appendice B

Seconda Appendice

Bibliografia

- [1] Renzo Davoli. Internet of Threads. <http://www.cs.unibo.it/~renzo/papers/2013.iciw.pdf>, 2013.
- [2] Renzo Davoli. Internet of Threads: Processes as Internet Nodes. <http://www.cs.unibo.it/~renzo/papers/2014.IntTechIoTh.pdf>, 2014.
- [3] Renzo Davoli. Internet of Threads. http://www.cs.unibo.it/~renzo/papers/ConfGARR11_SelectedPapers_Davoli.pdf.
- [4] Altran. picoTCP. <https://github.com/tass-belgium/picotcp>.
- [5] Virtual Square Team. vuos. <https://github.com/virtualsquare/vuos>.
- [6] Virtual Square Team. LWIPv6. <http://wiki.v2.cs.unibo.it/wiki/index.php/LWIPv6>.
- [7] Virtual Square Team. UMview. <http://wiki.v2.cs.unibo.it/wiki/index.php/UMview>.
- [8] Virtual Square Team. purelibc. <http://wiki.virtualsquare.org/wiki/index.php/PureLibc>.

- [9] NetLink associazione LUGMan (Linux Users Group Mantova). <http://lugman.org/images/4/43/NetLink.pdf>, 2009.
- [10] Thomas Graf. <https://www.infradead.org/~tgr/libnl/>.
- [11] Thomas Graf. <https://github.com/tgraf/libnl>.
- [12] Thomas Haller. <https://github.com/thom311/libnl>.
- [13] LWIP. <https://savannah.nongnu.org/projects/lwip/>.

Ringraziamenti

Qui possiamo ringraziare il mondo intero!!!!!!!!!!
Ovviamente solo se uno vuole, non è obbligatorio.