

Fondamenti dell'Architettura Internet e Vulnerabilità Intrinseche dei Protocolli IP/TCP

(Appunti elaborati a partire dalle slide del corso
Network and System Defense, A.A. 2025/2026)

Leonardo Polidori, Edoardo Marchionni, Chat GPT

6 novembre 2025

Indice

1 NET_01	6
1.1 Architettura di base e principi di rete	6
1.1.1 Definizione e interconnessione	6
1.1.2 Indirizzamento e instradamento	6
1.1.3 Anatomia dell'indirizzo IP e subnetting	6
1.1.3.1 1) Cosa significa “/27”	7
1.1.3.2 2) Perché basta guardare l'ultimo otetto	7
1.1.3.3 3) Trova il blocco in cui cade 99	7
1.1.3.4 4) Verifica con l'AND (ultimo otetto)	8
1.1.3.5 5) Broadcast: tutti i bit host a 1	8
1.1.3.6 6) Intervallo host e conteggio	8
1.1.4 Sistemi autonomi (AS) e routing globale	8
1.1.5 La routing table e l'algoritmo di lookup	8
1.2 II. Il viaggio del pacchetto: esempio di richiesta DNS	8
1.2.1 Topologia e hop	8
1.2.2 Stack protocollare e incapsulamento	9
1.3 III. Vulnerabilità intrinseche di IP e TCP	9
1.3.1 Identificazione, spoofing e non-ripudio	10
1.3.2 Confidenzialità	10
1.3.3 Integrità dei dati	10
1.3.4 Packet replication e anti-replay	10
1.3.5 Insicurezza delle mappature dinamiche	10
1.4 Laboratorio 1: MiTM e DNS spoofing	11
1.4.1 Obiettivo e scenario	11
1.4.2 Fasi dell'attacco	11
1.4.3 STEP 1: ARP spoofing (MiTM)	11
1.4.4 STEP 2 & 3: intercettazione e DNS spoofing	12
1.4.5 STEP 4: impersonificazione del sito web	12
2 NET_02	13
2.1 Sicurezza delle Reti Ethernet LAN (Livello 2)	13
2.2 Perché la LAN Ethernet è fragile per natura	13
2.2.1 Frame, indirizzamento e forwarding	13
2.2.1.1 Multiport Repeaters (Hub)	13
2.2.1.2 Bridge/Switches	13

2.2.1.3	Address Learning	13
2.2.2	Topologie e controllo dei loop: STP	14
2.2.3	Adattamento del Livello 3 su Livello 2: DHCP, ARP e NDP	15
2.2.3.1	DHCP — Dynamic Host Configuration Protocol (IPv4).	15
2.2.3.2	ARP — Address Resolution Protocol (IPv4).	16
2.2.3.3	NDP — Neighbor Discovery Protocol (IPv6).	16
2.2.4	Vulnerabilità e minacce principali	16
2.2.4.1	Accesso alla rete	16
2.2.4.2	Compromissione e controllo dello switch.	17
2.2.4.3	Riservatezza e intercettazione.	17
2.2.4.4	Integrità del traffico e attacchi Man-in-the-Middle (MITM)	18
2.2.4.5	Disponibilità e attacchi di Denial of Service (DoS)	18
2.2.5	Contromisure: dal minimo sindacale al robusto	19
2.2.5.1	Router-based security (segmentazione L3).	19
2.2.6	Controllo d'accesso (Access Control)	20
2.2.6.1	Obiettivo.	20
2.2.6.2	802.1X (port-based NAC).	20
2.2.6.3	Autorizzazione dinamica con 802.1X.	20
2.2.6.4	Port Security & ACL sugli switch.	20
2.2.6.5	Segmentazione con VLAN (complemento).	20
2.3	Laboratorio 2A: ACL L2 con Linux Bridge ed ebtables (binding MAC→porta) .	21
2.3.0.1	1) Preparazione host (MAC/IP dei client).	21
2.3.0.2	2) Creazione del bridge L2 (lato “switch” Linux).	21
2.3.0.3	3) ACL L2 con ebtables (NETFILTER).	22
2.4	Laboratorio 2B: MACsec su Linux (confidenzialità, integrità e anti-replay a L2) .	22
2.4.0.1	Host A (client1).	22
2.4.0.2	Host B (client2).	23
2.4.0.3	Opzioni di sicurezza e test.	23
2.4.1	Secure Address Resolution (IPv4/IPv6)	24
2.4.1.1	IPv4: dal piano di switching ai vincoli per-presa.	24
2.4.1.2	IPv4: proposte crittografiche.	24
2.4.1.3	IPv6: NDP sicuro.	24
2.4.2	Security Monitoring	24
2.4.2.1	Firewall & DPI.	24
2.4.2.2	IDS/IPS e accesso al traffico.	24
2.4.2.3	Dove il monitoring aiuta davvero.	24
2.4.2.4	Messaggio chiave (dalle slide).	24
3	NET_03 — Virtual LANs (VLAN)	25
3.0.1	Definizione e motivazione	25
3.0.2	Limiti delle reti fisicamente separate	25
3.0.2.1	Benefici principali delle VLAN	25
3.1	Assegnazione e membership delle VLAN	26
3.1.1	Criteri di assegnazione	26
3.1.2	Vista logica	26
3.1.2.1	Router “one-armed”	27
3.2	Trasporto dei frame: Tagging IEEE 802.1Q	28
3.2.1	Porte di tipo Access, Trunk e Hybrid	28
3.2.2	Access links	28
3.2.3	Access links nelle regioni legacy	29
3.2.4	Trunk links	29
3.2.5	Hybrid links	30

3.2.6	Una stazione può appartenere a più VLAN?	31
3.3	Laboratorio 3: Configurazione VLAN e router one-armed	33
3.3.0.1	Isolamento del traffico	34
3.3.0.2	Routing inter-VLAN	34
3.3.0.3	Verifica del comportamento	35
3.3.0.4	Osservazione pratica	35
3.4	Sicurezza delle VLAN e vulnerabilità di livello 2	36
3.4.1	Minacce principali	36
3.4.1.1	1) MAC Flooding (CAM Overflow)	36
3.4.1.2	2) ARP Spoofing / Poisoning	36
3.4.1.3	3) VLAN Hopping	37
3.4.1.4	4) Attacchi ai protocolli di controllo	37
3.5	Laboratorio 4: VLAN Hopping e Double Tagging	39
3.5.1	Scenario	39
3.5.2	Topologia del laboratorio	39
3.5.3	Prerequisiti perché l'attacco riesca	40
3.5.4	Funzionamento dell'attacco (passo–passo)	40
3.5.5	Esecuzione pratica (Linux)	40
3.5.5.1	Cosa osservare con lo sniffer	41
3.5.5.2	Limitazioni pratiche	41
3.5.5.3	Mitigazioni	41
4	NET_04 — 802.1x	42
4.1	Quadro generale e obiettivo	42
4.2	Attori dell'architettura (vista di alto livello)	42
4.3	EAP ed EAPoL: fondamenti	42
4.3.0.1	EAP (Extensible Authentication Protocol).	42
4.4	EAP termination vs EAP relay (EAPoR)	43
4.4.0.1	EAP termination mode (terminazione locale).	43
4.4.0.2	EAP relay mode (EAP over RADIUS, EAPoR).	43
4.5	Processi di autenticazione: dai diagrammi alle implicazioni operative	44
4.5.0.1	Message exchange (visione di alto livello).	44
4.5.0.2	Relay Mode (EAP-MD5) — sequenza dettagliata.	45
4.5.0.3	EAP-TLS: esempio più complesso.	46
4.6	Operazioni aggiuntive: ri-autenticazione, logout, timer	46
4.7	Autorizzazione post-autenticazione: VLAN, ACL, UCL	46
4.7.0.1	VLAN dinamiche.	46
4.7.0.2	ACL per-utente.	46
4.7.0.3	UCL (User Control List).	47
4.8	Vulnerabilità storiche e motivazione per MACsec	47
4.9	Lab 5 — 802.1X Port-Based Authentication & VLAN assignment	48
4.9.0.1	Obiettivo.	48
4.9.0.2	Topologia di rete.	48
4.9.0.3	Configurazione dello switch (Authenticator).	48
4.9.0.4	Server di autenticazione (FreeRADIUS).	49
4.9.0.5	Configurazione dei client (Supplicant).	49
4.9.0.6	Controllo del traffico L2.	49
4.9.0.7	Sequenza operativa.	49
4.9.0.8	Osservazioni e risultati.	50
4.9.0.9	Conclusioni.	50
4.10	MACsec e MKA (802.1AE + 802.1X-2010)	51
4.10.1	Perché MKA: il tassello mancante dopo 802.1X	51

4.10.2	Le chiavi in gioco e come si ottengono	51
4.10.2.1	CAK e CKN	51
4.10.2.2	ICK e KEK	51
4.10.2.3	SAK	52
4.10.3	Due modalità d'uso del CAK	52
4.10.3.1	Static CAK	52
4.10.3.2	Dynamic CAK	52
4.10.4	Cosa si scambiano i peer: il ciclo MKA	52
4.10.4.1	Annuncio, stato e priorità	52
4.10.4.2	Elezione del Key Server	52
4.10.4.3	Apertura dei Secure Channel	52
4.10.4.4	Vita della sessione	53
4.10.5	Collegamento con le policy 802.1X/NAC	53
4.11	Simple MKA lab with Linux	54
4.12	Laboratorio: MKA/MACsec su Linux (Static CAK)	54
4.12.1	Scopo	54
4.12.2	Topologia	54
4.12.3	Procedura	54
4.12.3.1	Generazione delle chiavi (uguali su <i>tutti</i> i peer)	54
4.12.3.2	Configurazione MKA su ciascun host	54
4.12.3.3	Avvio di MKA e creazione dell'interfaccia <code>macsec0</code>	55
4.12.3.4	Indirizzamento IP sulla <code>macsec0</code>	55
4.12.3.5	Test base di connettività	55
4.12.4	Verifiche	55
4.12.4.1	Log MKA	55
4.12.4.2	Stato interfaccia	55
4.12.5	Troubleshooting	55
4.12.6	Pulizia	55
4.12.7	Estensioni & domande d'esame	55
4.12.7.1	Dynamic CAK (accenno)	55
5	SYS_01 — Introduzione ai Security Frameworks	56
5.1	Quadro generale e obiettivo	56
5.2	Principi fondamentali	56
5.2.0.1	Confidenzialità	56
5.2.0.2	Integrità	56
5.2.0.3	Disponibilità	56
5.2.0.4	Autenticità e Autorizzazione	57
5.2.0.5	Accountability	57
5.2.0.6	Dipendibilità	57
5.3	Perché servono i Security Frameworks	58
5.4	Tre famiglie a confronto	58
5.4.1	FIPS 200 — Requisiti minimi per i sistemi federali	58
5.4.2	CIS Critical Security Controls — L'igiene prioritaria	58
5.4.3	ISO/IEC 27000 — ISMS e gestione del rischio	59
6	SYS_02 — Hardware Primer	60
6.1	Moore's Law e l'evoluzione del calcolo	60
6.2	Fine della crescita lineare e parallelismo	60
6.3	Pipeline e architettura superscalare	60
6.4	Branch Prediction: principi e strategie	60
6.4.0.1	Perché serve una predizione	61

6.4.0.2	Tipi di predizione	61
6.4.0.3	Contatore a 2 bit saturante	61
6.4.0.4	Predittori correlati e multilivello	61
6.4.0.5	Ottimizzazioni hardware	62
6.4.0.6	Importanza crescente della predizione	62
6.5	Simultaneous Multithreading (SMT)	63
6.6	Pipeline interna nei processori Intel Xeon	63
6.7	Gerarchia di memoria	64
6.7.0.1	Inclusività e mappatura	64
6.7.0.2	Sostituzione ed aggiornamento	66
6.8	Cache coherence nei multicore	67
6.8.0.1	Protocolli di coerenza	68
6.9	Protocolli MOESI e VI	69
6.9.0.1	Virtual vs. Physical Cache Indexing	70
6.10	Hardware Transactional Memory (HTM)	71
6.10.0.1	Principio di funzionamento	71
6.10.0.2	Casi di abort	72
6.11	DRAM e Refresh	72
6.11.0.1	Effetti del refresh	72

1 NET_01

1.1 Architettura di base e principi di rete

1.1.1 Definizione e interconnessione

Internet è un'*inter-rete*: un insieme di numerose sotto-reti eterogenee connesse tra loro. Le sotto-reti possono essere basate su tecnologie diverse al Livello 2 (L2) — ad esempio 802.11 (WiFi), 802.3 (Ethernet), tecnologie cellulari (3G/4G), fibra ottica o ADSL — ma comunicano grazie a uno stack di protocolli comune, implementato sopra i diversi livelli fisici e MAC, nella logica del paradigma OSI. Il protocollo di base che abilita l'interoperabilità è l'Internet Protocol (IP), affiancato da protocolli di livello superiore come TCP, UDP e protocolli applicativi (per es. DNS, HTTP).

1.1.2 Indirizzamento e instradamento

Ogni dispositivo in una rete IP possiede un identificatore numerico univoco: l'indirizzo IP (32 bit per IPv4, 128 bit per IPv6). Le sotto-reti sono connesse mediante router, dispositivi che effettuano l'inoltro (forwarding) dei pacchetti dalla sorgente alla destinazione seguendo regole presenti nelle tabelle di instradamento.

Il forwarding si basa sul principio del *Longest Prefix Match* (LPM): per decidere quale voce della routing table utilizzare si seleziona la corrispondenza con il prefisso di rete più lungo che include l'indirizzo di destinazione. L'inoltro può essere:

- *diretto*, quando la destinazione si trova nella stessa rete locale;
- *indiretto*, quando la destinazione è raggiungibile tramite un next hop (salto successivo).

Example: 192.168.0.5 and 192.168.0.6 on the same network

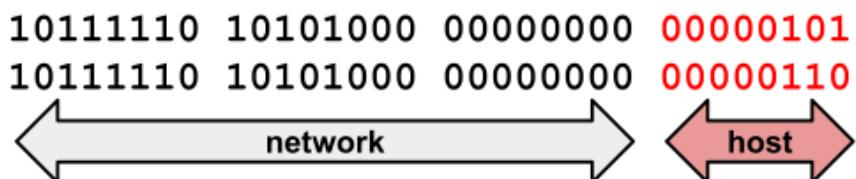


Figura 1: Instradamento diretto: sorgente e destinazione nella stessa rete locale (L2/L3).

Il compito di IP è consegnare il pacchetto alla rete finale; la consegna al dispositivo specifico è rimandata al livello L2, che si occupa della mappatura IP <-> L2 (per esempio tramite ARP per IP <-> MAC).

1.1.3 Anatomia dell'indirizzo IP e subnetting

Le reti IP sono suddivise in subnet logiche. Due host appartenenti alla stessa subnet condividono i primi X bit dell'indirizzo (la *parte di rete*), mentre i restanti $32 - X$ bit identificano l'host. Dal 1984 si usa CIDR (Classless Inter Domain Routing): il prefisso non è più implicito per classi fisse ma viene specificato tramite una *subnet mask* (o notazione “/length”), che indica quali bit rappresentano la parte di rete. L' i -esimo bit della subnet mask è settato a **0** se l' i -esimo bit è

nella host part; **1** se invece è nel prefisso network .Ad esempio, con indirizzo 192.168.1.12 e maschera 255.255.255.0 (ovvero /24) la rete è 192.168.1.0.

Example

<input type="checkbox"/> IP address: 192.168.1.12	10111110 10101000 00000001 00001100
<input type="checkbox"/> Network Mask: 255.255.255.0 (aka /24)	11111111 11111111 11111111 00000000

Network Prefix (address AND mask)
192.168.1.0

Figura 2: Esempio network prefix

Ogni rete ha due indirizzi ip **RISERVATI** ovvero:

- *Net address* tutti i bits nella parte host sono 0.
- *Broadcast address* tutti i bits nella parte host sono 1.

Example : find the network and broadcast addresses of host 209.85.129.99/27

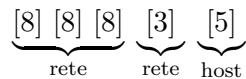
209.85.129.99 (IP addr host)	11010001 01010101 10000001 01100011
255.255.255.224 (Subnet Mask)	11111111 11111111 11111111 11100000
209.85.129.96 (IP addr network)	11010001 01010101 10000001 01100000
209.85.129.127 (IP addr broadcast)	11010001 01010101 10000001 01111111

Figura 3: trovare il network e broadcast address

Esempio spiegato: trovare rete e broadcast di 209.85.129.99/27

Vogliamo ricavare l'indirizzo di **rete** e di **broadcast** per l'host 209.85.129.99/27.

1.1.3.1 1) Cosa significa “/27” La notazione /27 dice che i **primi 27 bit** dell'indirizzo sono di *rete* e i rimanenti sono di *host*. Nel formato “a ottetti”, i primi 24 bit coincidono con i primi tre ottetti; quindi la /27 “entra” nel **quarto ottetto**:



La maschera corrispondente è 255.255.255.224, perché nell'ultimo ottetto i 3 bit di rete valgono $11100000_2 = 224$.

1.1.3.2 2) Perché basta guardare l'ultimo ottetto Con /27 i primi tre ottetti (209.85.129) restano identici per rete/host/broadcast. Tutta la partizione in sottoreti avviene nel **quarto ottetto**. Qui rimangono **5 bit di host** \Rightarrow ogni sottorete ha $2^5 = 32$ indirizzi contigui (un “blocco”).

1.1.3.3 3) Trova il blocco in cui cade 99 I blocchi nel quarto ottetto sono a passi di 32: 0-31, 32-63, 64-95, 96-127, 128-159, ... Poiché 99 appartiene a 96-127, la nostra **rete** è 209.85.129.96 e il **broadcast** sarà l'ultimo del blocco, 209.85.129.127. Verifichiamo con l'AND bit-a-bit.

1.1.3.4 4) Verifica con l'AND (ultimo ottetto) .

```
[basicstyle=\ttfamily\small,frame=single]
99 = 01100011
224 = 11100000 (maschera /27 nell'ultimo ottetto)
AND -----
01100000 = 96 --> indirizzo di rete (quarto ottetto)
```

Quindi l'indirizzo di rete è 209.85.129.96.

1.1.3.5 5) Broadcast: tutti i bit host a 1 Nel broadcast si *mantengono* i 3 bit di rete e si mettono a 1 i 5 bit di host:

$$\text{rete (ult. ottetto)} = 01100000 + 00011111 (\text{tutti i bit host a 1}) = 01111111 = 127.$$

Dunque broadcast = 209.85.129.127.

1.1.3.6 6) Intervallo host e conteggio Gli host validi sono i numeri compresi *tra* rete e broadcast:

$$209.85.129.97 \text{ fino a } 209.85.129.126,$$

per un totale di $2^5 - 2 = 30$ host (si escludono rete e broadcast).

1.1.4 Sistemi autonomi (AS) e routing globale

L'inter-rete è organizzata in Autonomous Systems (AS), domini amministrativi che gestiscono internamente le proprie politiche di instradamento. All'interno di un AS si adottano Interior Gateway Protocols (IGP) come OSPF, IS-IS o RIP; lo scambio di rotte tra AS diversi avviene tramite l'Exterior Gateway Protocol più usato: BGP, che garantisce la raggiungibilità globale.

1.1.5 La routing table e l'algoritmo di lookup

La routing table contiene entry costituite tipicamente da: indirizzo di destinazione, maschera/-netmask, next hop e interfaccia d'uscita. La funzione di lookup per un pacchetto p itera le voci ordinate per lunghezza del prefisso e restituisce la voce i tale che

$$(p.daddr \& i.mask) = i.addr,$$

dove $\&$ è l'AND bit-a-bit; se non si trova alcuna corrispondenza il pacchetto viene scartato.

1.2 II. Il viaggio del pacchetto: esempio di richiesta DNS

1.2.1 Topologia e hop

Un pacchetto generato da un browser per risolvere un nome (ad esempio `www.google.com`) percorre una serie di hop: rete domestica (WiFi), access point/router, edge router dell'AS dell'utente, router di confine (border router), una sequenza di AS di transito e infine il data center che ospita il servizio DNS o il server web. Ogni tratto può utilizzare tecnologie e politiche diverse, e rappresenta un potenziale punto di vulnerabilità.

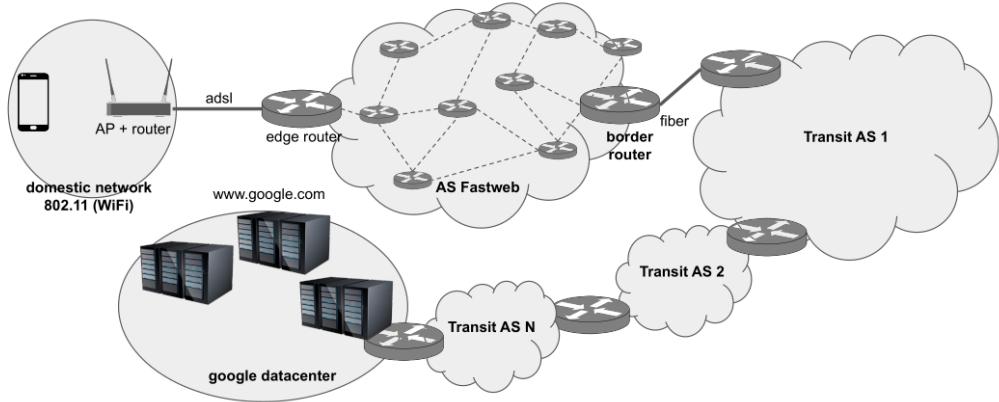


Figura 4: dal web browser al web server

1.2.2 Stack protocollare e encapsulamento

La richiesta DNS attraversa gli strati della pila:

- **Livello applicativo (DNS)**: genera la query di risoluzione ("dammi l'indirizzo ip di www.google.com").
- **Livello trasporto (UDP)**: aggiunge porta sorgente (es. 5000) e porta destinazione (53), oltre al checksum.
- **Livello rete (IP)**: inserisce indirizzi IP sorgente e destinazione (es. 10.0.0.100 e 85.18.200.200), TTL, eventuale fragmentation.
- **Livello accesso (L2)**: encapsula il frame con indirizzi MAC del next hop; questi cambiano ad ogni salto.

Durante il percorso gli indirizzi IP rimangono costanti, mentre gli indirizzi MAC vengono aggiornati hop-by-hop. Meccanismi come ARP permettono la traduzione dinamica IP <-> MAC all'interno di una stessa rete locale.

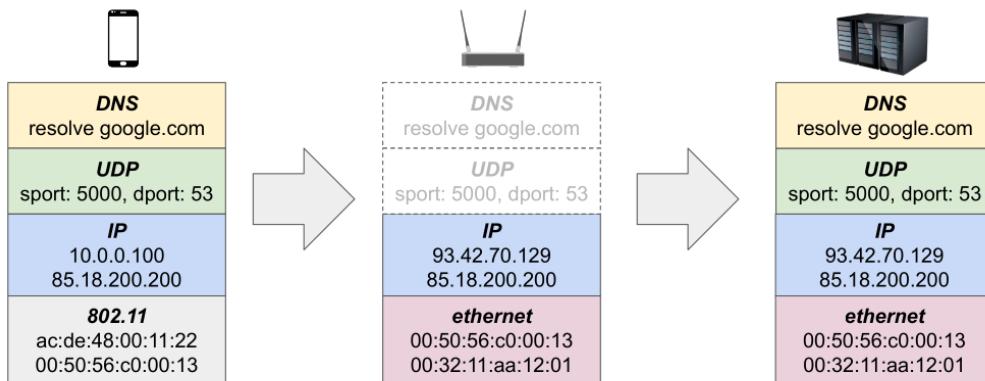


Figura 5: richiesta DNS (semplificata)

1.3 III. Vulnerabilità intrinseche di IP e TCP

I protocolli storici di Internet sono stati progettati principalmente per interoperabilità e scalabilità, non per la sicurezza. Questo ha lasciato diverse debolezze intrinseche.

1.3.1 Identificazione, spoofing e non-ripudio

Gli **identificatori di rete** (indirizzi IP e MAC) sono semplici stringhe binarie facilmente manipolabili: un mittente può generare pacchetti con sorgente falsata (**IP spoofing**) oppure modificare l'indirizzo sorgente di pacchetti che sta inoltrando. Questo fenomeno rende possibile, ad esempio, l'impersonificazione di server legittimi (un attaccante può inviare pacchetti che sembrano provenire da un DNS server affidabile). **IP non fornisce meccanismi di autenticazione dell'origine**: non esiste un modo intrinseco per dimostrare che l'indirizzo sorgente di un pacchetto corrisponda realmente al mittente fisico, provocando problemi di *repudiation*.

1.3.2 Confidenzialità

Il protocollo IP non cifra il payload né fornisce protezione contro l'intercettazione: catturare e decodificare pacchetti su un segmento di rete è, in molti casi, semplice. Inoltre gli utenti non controllano l'intero percorso seguito dai pacchetti; anche fidandosi del proprio ISP, sono necessari fiducia e verifiche su tutti gli AS attraversati. Attacchi di route hijacking o route leaking possono alterare il percorso e compromettere la riservatezza.

1.3.3 Integrità dei dati

IP, TCP e UDP usano checksum per rilevare errori di trasmissione (header e payload), ma questi meccanismi non sono progettati come primitive di sicurezza: sono vulnerabili a manipolazioni intenzionali poiché basta ricalcolare il checksum dopo la modifica del pacchetto. Per esempio, il checksum IP è una semplice somma/XOR su parole dell'header e non offre garanzie contro un attaccante attivo.

1.3.4 Packet replication e anti-replay

A livello IP **non** esistono numeri di sequenza o marcatori univoci che identifichino inequivocabilmente un pacchetto in un flusso; il problema anti-replay è quindi in gran parte non risolto a questo livello. TCP fornisce numeri di sequenza, ma essi sono destinati alla gestione dell'affidabilità e dell'ordine, non all'autenticazione. Poiché tali numeri non sono protetti criptograficamente, possono essere predetti o spoofati in alcuni scenari, consentendo replay o session hijacking se non vengono adottate contromisure a livello superiore (ad esempio TLS).

1.3.5 Insicurezza delle mappature dinamiche

Molti servizi critici si basano su mappature dinamiche non progettate per la sicurezza: DNS (nomi→IP), ARP (IP→MAC), tabelle di bridging (MAC→porta), e la stessa routing table (destinazione→next hop). Implementazioni legacy, come il DNS non autenticato, permettono a un attaccante di fornire risposte fasulle: la risoluzione nome→IP non è intrinsecamente verificabile senza meccanismi come DNSSEC.

Laboratorio

1.4 Laboratorio 1: MiTM e DNS spoofing

1.4.1 Obiettivo e scenario

L'obiettivo è dirottare richieste HTTP non cifrate attraverso DNS spoofing e impersonificazione di un sito (es. <http://netgroup.uniroma2.it>). Lo scenario tipico prevede un attaccante nella stessa rete locale della vittima; il resolver della vittima è configurato su un DNS pubblico (per es. 8.8.8.8). L'attacco combina:

1. lo sfruttamento della mappatura IP <-> MAC (via ARP spoofing) per stabilire un MiTM;
2. la manipolazione delle risposte DNS per risolvere il nome del sito bersaglio verso un indirizzo controllato dall'attaccante.

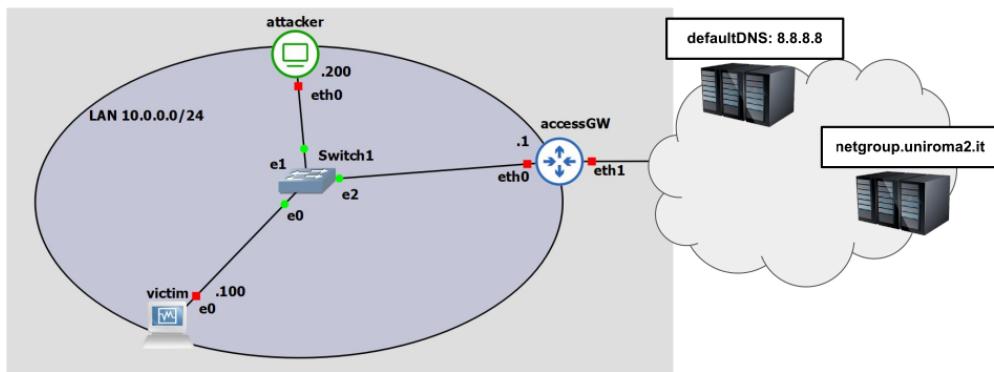


Figura 6: Topologia rete

1.4.2 Fasi dell'attacco

L'attacco tipico è articolato in quattro step principali:

1. **STEP 1 – MiTM (ARP spoofing):** l'attaccante dissipa nelle cache ARP della vittima e del gateway risposte ARP falsificate in modo da farsi passare per entrambi e intercettare il traffico.
2. **STEP 2 – Intercettazione della richiesta DNS:** una volta in posizione di MiTM, l'attaccante intercetta le query DNS emesse dalla vittima.
3. **STEP 3 – Spoofing della risposta DNS:** l'attaccante risponde con una risoluzione falsa per il dominio bersaglio, puntando a un IP di sua proprietà.
4. **STEP 4 – Impersonificazione del sito web:** l'attaccante serve una copia del sito (ottenuta tramite mirroring) dall'IP di controllo, così che la vittima riceva contenuti apparentemente legittimi.

1.4.3 STEP 1: ARP spoofing (MiTM)

L'attaccante invia risposte ARP non richieste (opcode 2) sia alla vittima che al default gateway:

- alla vittima: un frame ARP unicast indirizzato al MAC della vittima (`bb:bb:bb:bb:bb:bb`) affermando che l'IP del router (10.0.0.1) corrisponde al MAC dell'attaccante (`aa:aa:aa:aa:aa:aa`);

- al gateway: un frame ARP unicast indirizzato al MAC del router (`cc:cc:cc:cc:cc:cc`) affermando che l'IP della vittima (10.0.0.100) corrisponde al MAC dell'attaccante (`aa:aa:aa:aa:aa:aa`).

Ripetendo queste risposte periodicamente, l'attaccante mantiene la posizione di MiTM.

1.4.4 STEP 2 & 3: intercettazione e DNS spoofing

Dopo aver stabilito il MiTM, l'attaccante può reindirizzare le richieste DNS verso la sua macchina:

Listing 1: Esempio: regola iptables per reindirizzare richieste DNS (UDP 53) alla macchina locale

```
iptables -t nat -A PREROUTING -p udp --dport 53 -j REDIRECT
```

Sulla macchina dell'attaccante viene eseguito un server DNS leggero (es. `dnsmasq`) con una configurazione del tipo:

Listing 2: Estratto di /etc/dnsmasq.conf

```
interface=eth0
no-dhcp-interface=eth0
server=1.1.1.1

# Risolvi il dominio bersaglio verso l'IP dell'attaccante
address=/netgroup.uniroma2.it/10.0.0.200
```

Questa configurazione restituisce per `netgroup.uniroma2.it` l'indirizzo 10.0.0.200; tutte le altre query vengono inoltrate al resolver pubblico (qui 1.1.1.1).

1.4.5 STEP 4: impersonificazione del sito web

L'attaccante può aver replicato il contenuto del sito bersaglio tramite strumenti di mirroring, ad esempio:

```
wget --mirror --convert-links --html-extension --no-parent -l 1 \
--no-check-certificate http://netgroup.uniroma2.it
```

I contenuti mirrorati vengono serviti localmente (per es. con Apache2). Così la vittima, ricevendo l'IP dell'attaccante per il dominio richiesto, ottiene una copia apparentemente autentica del sito.

Conclusione e contromisure (sintesi)

Le vulnerabilità descritte evidenziano che senza meccanismi di autenticazione, integrità e confidenzialità a livello superiore, l'infrastruttura IP/TCP è esposta a compromissioni. Contromisure pratiche includono:

- utilizzo diffuso di canali cifrati e autenticati (TLS/HTTPS) per proteggere le applicazioni;
- adozione di estensioni e protocolli progettati per la sicurezza (es. DNSSEC per autenticare risposte DNS, IPsec per integrità/confidenzialità a livello IP dove applicabile);
- tecniche di difesa a livello di rete locale (ARP inspection, dynamic ARP protection, filtraggio di pacchetti spoofati sui router e access control lists);
- pratiche operative: aggiornamento dei software, monitoraggio delle anomalie di routing e validazione delle rotte BGP.

2 NET_02

2.1 Sicurezza delle Reti Ethernet LAN (Livello 2)

2.2 Perché la LAN Ethernet è fragile per natura

Ethernet nasce per *autoconfigurarsi*: gli switch imparano indirizzi e percorsi (MAC learning), i protocolli di controllo (STP, ARP, DHCP) si basano su broadcast e sulla mancanza di autenticazione a L2. Questo rende immediati tre vettori: **osservare** (eavesdropping), **manipolare** (spoofing/MITM) e **interrompere** (DoS). Le minacce si organizzano in quattro famiglie: (i) accesso a rete/sistemi, (ii) confidenzialità, (iii) disponibilità, (iv) integrità.

2.2.1 Frame, indirizzamento e forwarding

Gli indirizzi Mac sono a 48 bit.

nella versione originale dell'ethernet la tipologia del frame veniva usata per il demultiplexing del layer superiore per esempio 0x800=IP; mentre in 802.3 indica la lunghezza oppure il tipo in particolare se il frame supera i $0x0600$ (1536_{10}) allora indica la tipologia di frame altrimenti LLC per il demultiplexing e indica il payload size. Se frame inferiore ai 46 bit allora viene utilizzato del padding.

Il primo bit dell'indirizzo MAC indica se l'indirizzo è unicast (0) o di gruppo/multicast (1); il secondo bit distingue tra indirizzi globali (0, assegnati dal produttore) e locali (1, configurabili via software o driver).

2.2.1.1 Multiport Repeaters (Hub) Gli hub, o ripetitori multiporta, operano come un bus condiviso: tutto il traffico ricevuto viene rigenerato su tutte le porte. Appartengono quindi a un unico **dominio di collisione** e non offrono isolamento tra host; per questo sono oggi sostituiti dagli switch.

2.2.1.2 Bridge/Switches :

Gli switch possono operare in:

- **Store&Forward**: lettura completa del frame (memorizzazione su buffer), controllo CRC, scarto dei frame *runt* (<64 bytes too short)/troppo lunghi oppure se fallisce il CRC; look up nella tabella e forwarding.
- **Cut-through**: lettura solo fino all'indirizzo, nessun check di integrità, look-up, forwarding.

Il **Forwarding Database** (FDB) mappa MAC→porta; le entry dinamiche sono apprese (MAC learning) e scadono con *ageing* tipicamente nell'ordine dei 300 s; altrimenti impostate staticamente da un sysadmin o da un db. Se la destinazione è sconosciuta, lo switch effettua *flooding*.

2.2.1.3 Address Learning Un frame arriva alla porta X quindi deve provenire dalla LAN connessa dalla porta X, il source address viene usato per l update del forwarding DB. Se arriva un frame da un source addr non presente nella tabella allora viene creata la entry con *age* = 0; se invece arriva un entry già presente viene refreshata la age di quella entry.

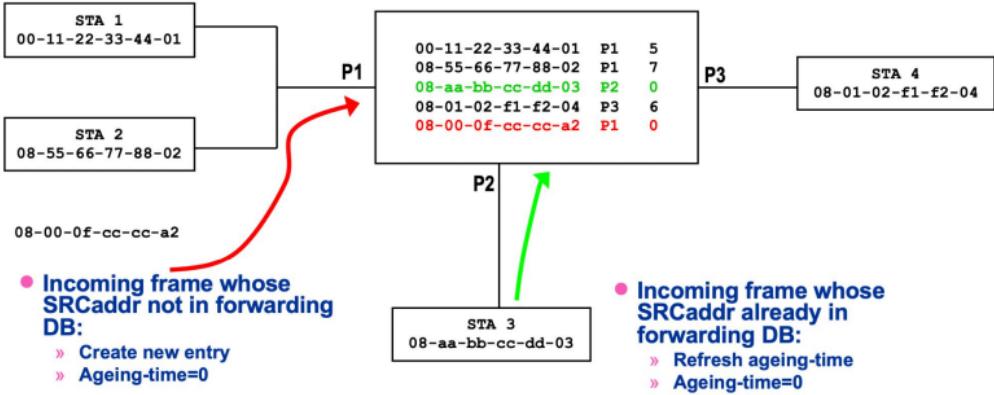


Figura 7: Topologia rete

Infine se arriva un frame da un source addr già presente nella tabella ma sotto una porta differente allora viene aggiornata la entry e refresh della age.

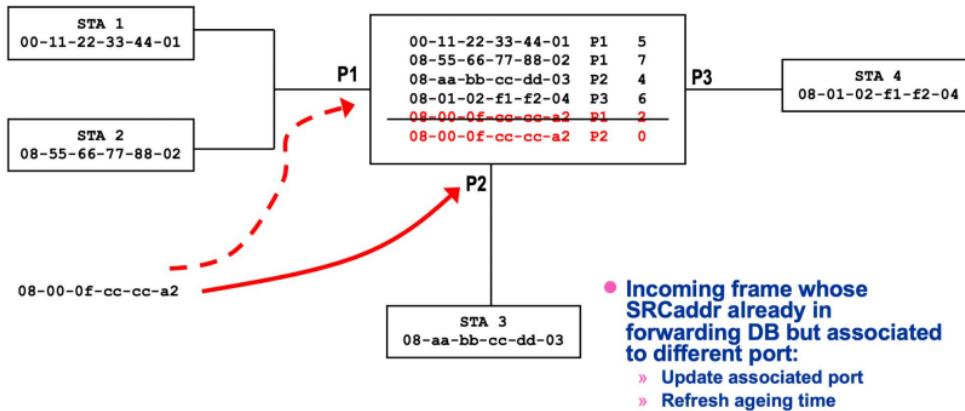


Figura 8: Topologia rete

2.2.2 Topologie e controllo dei loop: STP

Quando in una rete Ethernet esistono collegamenti ridondanti, possono formarsi *loop* a **Livello 2 (L2, Data Link)**: i frame possono circolare all'infinito tra switch, saturando la rete. Per evitare questo, entra in gioco lo **Spanning Tree Protocol**. L'idea è eleggere uno *switch radice* (**root bridge**) e disattivare alcune porte in modo che la topologia effettiva sia un albero evitando così loop, pur lasciando i link ridondanti pronti a subentrare in caso di guasti.

Gli switch si mettono d'accordo scambiandosi messaggi di controllo chiamati **BPDU (Bridge Protocol Data Unit)**. Ogni BPDU contiene l'identità dello switch (*Bridge ID*, che include *priorità* e MAC) e i *costi* dei percorsi. Viene eletto un *root bridge*; e poi per ogni altro switch, **STP** sceglie:

- una **porta radice (root port)**: la porta verso il root bridge con costo minore;
- eventuali porte in eccesso vengono messe in stato **bloccato (blocking)** per rompere i cicli.

Il risultato è che solo alcune porte sono di forwarding, mentre le altre restano **bloccate**; se un link o uno switch si guastano allora STP riconfigura la topologia evitando nuovamente loop.

Sul piano della sicurezza, però, STP ha un limite: a L2 non c'è **autenticazione** dei messaggi di controllo. Un host malevolo può inviare **BPDU** artefatto e farsi eleggere *root bridge* (alzando la priorità o manipolando i costi), dirottando o interrompendo il traffico. Per questo, in produzione si usano contromisure come *BPDU Guard*, *Root Guard*, *portfast* solo sugli host, e piani di controllo isolati.

2.2.3 Adattamento del Livello 3 su Livello 2: DHCP, ARP e NDP

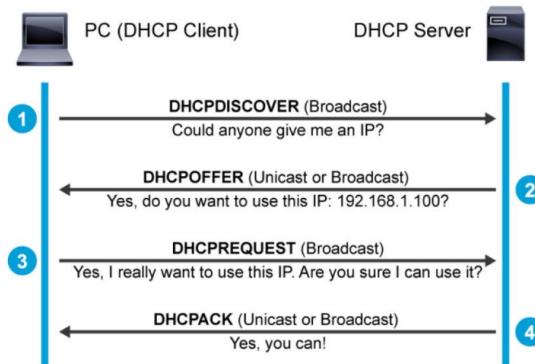
L'interoperabilità tra il **Livello 2 (Data Link)** e il **Livello 3 (Network)** è assicurata da una serie di protocolli di adattamento che consentono agli host di configurare automaticamente i propri parametri IP e di risolvere gli indirizzi fisici (MAC) dei dispositivi vicini. Questi protocolli, fondamentali per il funzionamento delle reti IP, nascono però in un contesto di fiducia implicita e **assenza di autenticazione**, diventando quindi bersagli ideali per attacchi di spoofing e manipolazione del traffico.

2.2.3.1 DHCP — Dynamic Host Configuration Protocol (IPv4). Il **DHCP** automatizza l'assegnazione degli indirizzi IP e dei parametri di rete,. Opera in modalità *client-server* e sfrutta il meccanismo di broadcasting a livello Ethernet per raggiungere il server anche quando il client non ha ancora un IP.

Il protocollo segue un tipico **handshake a quattro fasi**:

1. **DHCP Discover** — il client trasmette in broadcast (255.255.255.255) per cercare un server disponibile;
2. **DHCP Offer** — il server risponde offrendo un indirizzo IP e altri parametri (gateway, DNS, lease time);
3. **DHCP Request** — il client accetta esplicitamente un'offerta specifica;
4. **DHCP ACK** — il server conferma l'assegnazione e crea una voce di *lease* nel proprio database.

Ogni lease è temporaneo e può essere rinnovato tramite messaggi *DHCP Renew/Rebind*. Quando un host si trova in una rete diversa dal server, la comunicazione avviene tramite un **DHCP relay agent** — spesso un router — che incapsula i messaggi Discover/Request e li inoltra verso il server remoto, mantenendo così la visibilità dell'origine (campo *giaddr*).



Sicurezza: DHCP non autentica né il client né il server. Un host malevolo può rispondere più velocemente del server legittimo (**DHCP spoofing**) e assegnare gateway o DNS controllati,

dirottando il traffico. Per mitigare questi scenari, gli switch moderni implementano **DHCP Snooping**: una funzione che registra le associazioni IP–MAC–porta–VLAN apprese dai messaggi DHCP legittimi, utile anche per alimentare altri controlli di sicurezza come la *Dynamic ARP Inspection*.

2.2.3.2 ARP — Address Resolution Protocol (IPv4). L'ARP consente di scoprire l'indirizzo MAC associato a un indirizzo IP all'interno della stessa LAN. Quando un host deve inviare un pacchetto IP verso una destinazione della propria subnet, interroga la rete inviando un messaggio **ARP Request** in broadcast contenente l'indirizzo IP cercato. L'host corrispondente risponde con un **ARP Reply** unicast, fornendo il proprio MAC address. Ogni sistema mantiene una **cache ARP** che memorizza temporaneamente queste associazioni per ridurre il numero di richieste future (tipicamente 20 minuti su sistemi UNIX-like).

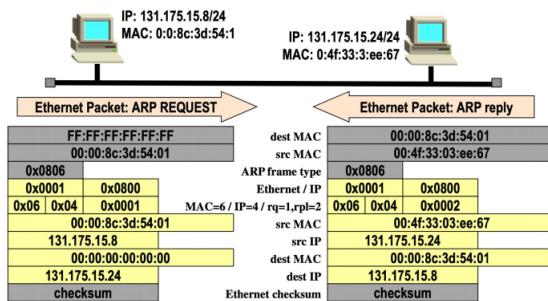


Figura 9

Debolezze: ARP è *stateless* e non prevede autenticazione. Un attaccante può quindi inviare **ARP Reply falsificati** (anche senza richiesta) per associare un IP legittimo al proprio MAC address: è il classico **ARP poisoning**, che consente di intercettare, modificare o bloccare il traffico (attacco *Man-in-the-Middle*). Meccanismi come **Dynamic ARP Inspection (DAI)**, basati sulle tabelle di DHCP Snooping, permettono di bloccare risposte ARP incoerenti rispetto alle associazioni IP–MAC note.

2.2.3.3 NDP — Neighbor Discovery Protocol (IPv6). Nel mondo IPv6, il **Neighbor Discovery Protocol (NDP)**, sostituisce ARP e parte del ruolo di DHCP. Basato su **ICMPv6** A differenza di ARP, NDP usa **multicast** anziché broadcast, riducendo l'impatto sulla rete e migliorando la scalabilità. Tuttavia, condivide la stessa assenza di autenticazione: un host può fingere di essere un router o un vicino legittimo, portando a *neighbor spoofing* o *router advertisement flooding*.

2.2.4 Vulnerabilità e minacce principali

La sicurezza di una rete Ethernet dipende fortemente dall'affidabilità del Livello 2, ma i protocolli su cui si basa — come ARP, DHCP e STP — sono stati progettati per un ambiente fidato, senza autenticazione o cifratura. Questo rende l'intera architettura LAN intrinsecamente vulnerabile ad attacchi che puntano al controllo dell'accesso fisico, alla manipolazione del traffico e al degrado delle prestazioni. Le principali minacce si raggruppano in quattro categorie: **(1) accesso alla rete, (2) riservatezza del traffico, (3) integrità e manipolazione, (4) disponibilità e prestazioni**.

2.2.4.1 Accesso alla rete .

Accesso fisico non autorizzato. L'attacco più basilare consiste nel collegarsi fisicamente a una porta Ethernet lasciata attiva e non monitorata. In ambienti non presidiati (aula, uffici, open space), un attaccante può semplicemente inserire il proprio dispositivo o un piccolo switch/access point (*rogue device*), espandendo la rete interna senza autorizzazione. Poiché lo standard Ethernet non prevede autenticazione a livello di porta, la connessione risulta immediatamente operativa. Questo tipo di “*join fisico*” rappresenta il punto d'ingresso di molte compromissioni LAN.

Accesso remoto e ricognizione. Una volta ottenuto l'accesso (fisico o logico), l'attaccante può mappare la topologia di rete sfruttando protocolli di base:

- *ARP scanning* — per enumerare gli indirizzi IP attivi nella LAN;
- richieste *DHCP* — per dedurre l'intervallo di indirizzi disponibili e i parametri di rete (gateway, DNS);
- *port scanning* — per identificare i servizi esposti e i sistemi operativi in uso.

Queste informazioni consentono di costruire una mappa logica della rete e di selezionare successivi obiettivi di attacco (*target profiling*).

2.2.4.2 Compromissione e controllo dello switch. Gli switch possono diventare obiettivi diretti se il *management plane* è esposto o debolmente protetto. Molti dispositivi arrivano con **credenziali di default** (o addirittura senza password) e spesso prevedono un **reset fisico** che ripristina i parametri di fabbrica: in entrambi i casi un attaccante può ottenere l'accesso amministrativo. Una volta dentro, è possibile **dirottare il traffico** abbassando link strategici, **farsi eleggere root bridge** in *Spanning Tree* aumentando la priorità dello switch, o **indurre DoS** su link selezionati. Inoltre, a seconda dei *protocolli di gestione* abilitati, l'attaccante può **attivare il port mirroring** per intercettare i flussi e (*in alcuni scenari*) ottenere visibilità o accesso a VLAN non previste.

Dal lato protocolli L2, l'assenza di autenticazione a livello Ethernet rende possibile **manipolare STP** (fingendosi switch legittimo) per alterare la topologia o causare riconvergenze continue: è la base di diversi vettori di *denial-of-service*.

2.2.4.3 Riservatezza e intercettazione. **Eavesdropping (intercettazione).** In una rete Ethernet tradizionale basata su hub, tutto il traffico viene propagato su tutte le porte, rendendo facile l'intercettazione passiva (*sniffing*). Anche negli switch moderni, l'attacco resta possibile tramite:

- installazione fisica di un dispositivo di ascolto (*tap*) su un cavo in rame o fibra ottica;
- abilitazione di una scheda di rete in *promiscuous mode* per ricevere frame non destinati al proprio MAC (annulla filtering MAC);
- abuso della funzione di *port mirroring* sugli switch, utile per intercettare il traffico di altre porte.

MAC flooding. Gli switch memorizzano le associazioni MAC→porta nella CAM (Content Addressable Memory) o FDB (Forwarding Database). Un attaccante può saturare questa memoria inviando migliaia di frame con indirizzi MAC sorgente casuali. Quando la tabella si riempie, lo switch passa alla modalità *flooding*, inoltrando i frame su tutte le porte come un hub. Questo permette di catturare traffico unicast normalmente privato e, in certi casi, di rompere l'isolamento tra VLAN (*cross-VLAN leakage*).

MAC spoofing. Manipolando l'indirizzo MAC sorgente dei frame inviati, un attaccante può sostituirsi a un host legittimo già presente nella FDB. Il risultato è un *hijacking* del traffico diretto alla vittima, che può essere intercettato, modificato o semplicemente bloccato. Lo spoofing è

efficace perché Ethernet non verifica la coerenza tra il MAC dichiarato nel frame e quello della scheda di rete.

2.2.4.4 Integrità del traffico e attacchi Man-in-the-Middle (MITM)

ARP poisoning. Approfittando del fatto che l'ARP accetta qualunque risposta non autenticata, e anche quelle non richieste, un attaccante può inviare *ARP replies* falsificati per associare il proprio MAC address all'indirizzo IP di un altro nodo (ad esempio il gateway). In questo modo intercetta tutto il traffico tra la vittima e il router, realizzando un classico attacco *Man-in-the-Middle (MITM)*. Varianti di questo attacco esistono anche in IPv6 sotto forma di *Neighbor Advertisement spoofing* contro NDP.

DHCP poisoning. Simile nel principio, l'attacco DHCP poisoning consiste nel rispondere alle richieste DHCP più rapidamente del server legittimo. Il client riceve così parametri falsi (indirizzo IP, gateway, DNS), che permettono all'attaccante di deviare il traffico verso sistemi controllati o intercettarlo.

Session hijacking. Una volta intercettato il traffico (via ARP o DHCP poisoning), è possibile analizzare le sessioni di livello superiore (ad esempio TCP) e riprodurle, utilizzando numeri di sequenza o cookie d'autenticazione per impersonare un utente o un servizio.

Replay attack. Consiste nel riutilizzare pacchetti validi intercettati in precedenza — ad esempio messaggi di controllo o di autenticazione — per ottenere accesso o indurre comportamenti anomali nei dispositivi di rete. In mancanza di firme digitali o timestamp, questi messaggi vengono accettati come legittimi.

2.2.4.5 Disponibilità e attacchi di Denial of Service (DoS)

STP DoS e manipolazione topologica Il protocollo **Spanning Tree Protocol** non prevede autenticazione dei messaggi di controllo (**BPDU**, **Bridge Protocol Data Units**). Un attaccante può sfruttare questa debolezza per:

- eleggersi come *root bridge* forzando il traffico a passare attraverso il proprio nodo;
- inondare la rete di BPDU falsi, causando continui ricalcoli dell'albero di spanning e interruzioni periodiche dei collegamenti (*STP reconvergence loops*).

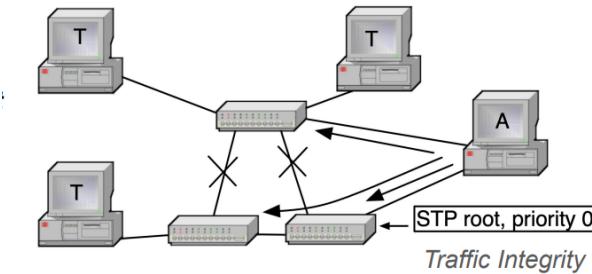


Figura 10

Resource exhaustion e flooding. Un altro vettore comune consiste nel sovraccaricare il piano di controllo degli switch o dei router. Attacchi di *unknown-unicast flooding* e tempeste di broadcast (*broadcast storms*) possono saturare la banda o la memoria, impedendo il corretto

forwarding dei frame. Simili risultati possono essere ottenuti generando una quantità eccessiva di richieste ARP o DHCP, portando al collasso del servizio (*DoS a livello L2*).

2.2.5 Contromisure: dal minimo sindacale al robusto

Per affrontare le vulnerabilità strutturali di Ethernet e le minacce che ne derivano, è fondamentale adottare un approccio stratificato. Le contromisure spaziano da soluzioni basilari, come la segmentazione della rete, a soluzioni avanzate che includono l'uso di crittografia L2 e l'autenticazione a livello di porta.

2.2.5.1 Router-based security (segmentazione L3). Sostituire (o affiancare) lo switch centrale con un **router IP** spezza la LAN in più *segmenti L2* separati (domini di broadcast distinti). Questo ha due effetti chiave:

1. **blocca i protocolli di controllo L2** e tutto il traffico broadcast/multicast tra segmenti (ARP, STP, DHCP non attraversano il confine L3 a meno di funzioni dedicate),
2. rende **impossibili gli attacchi L2 tra segmenti** (eavesdropping, MAC-table attacks, VLAN hopping, MITM via ARP/DHCP) perché gli header MAC si fermano al router e l'instradamento avviene sul piano IP. Le stesse minacce restano possibili *all'interno* di ciascun segmento se questo continua a essere uno switch L2.

Perché funziona. Il confine L3 elimina la connettività a livello Ethernet tra i segmenti: un host in segmento A non può inviare ARP o BPDU verso segmento B, né può “floodare” unicast sconosciuti oltre il router. Di conseguenza, **ARP spoofing**, **MAC flooding**, manipolazioni **STP** e **VLAN hopping** non propagano tra segmenti diversi. Il traffico inter-segmento è visibile solo sul router (o firewall), dove si possono applicare ACL e ispezioni più robuste.

Listing 3: Schema: ARP confinato dal confine L3 (il router non inoltra ARP)

```
[Host A 10.0.10.23] -- (Switch L2 VLAN10) -- [Router L3] -- (Switch L2 VLAN20) -- [Host B 10.0.20.45]

Domanda: A vuole parlare con 10.0.20.45 -> prima deve risolvere MAC.
- ARP Request "Chi ha 10.0.20.45?" \e un BROADCAST L2 (ff:ff:ff:ff:ff:ff).
- Il BROADCAST rimane nel DOMINIO L2 (VLAN10). Il router L3 NON inoltra ARP.
- A ARP-a solo il suo gateway (10.0.10.1) per uscire dal segmento; il resto \e routing IP.
Conclusione: ARP spoofing/flooding resta confinato entro la VLAN/segmento L2.
```

Listing 4: Schema: DHCP con relay (giaddr) oltre il confine L3

```
Client (senza IP) Switch/VLAN10 Router L3 (DHCP Relay) DHCP Server (172.16.0.10)

0.0.0.0:68 --[BCAST]--> DISCOVER --[riceve]--> aggiunge giaddr=10.0.10.1 --[UNICAST UDP/67]--> Server
                                                <---[UNICAST UDP/67]--- OFFER (
                                                dest: 10.0.10.1)
Relay inoltra in VLAN10 come BROADCAST verso il client (UDP/68)
Client --[BCAST]--> REQUEST --> Relay --[UNICAST]--> Server
                                                <---[UNICAST]--- ACK (lease, GW,
                                                DNS)
Relay inoltra l'ACK in VLAN10 (di solito come broadcast) al client.
Esito: lease IP (es. 10.0.10.50), gateway 10.0.10.1, DNS..., registrati dal server.
Nota: senza relay, i messaggi DHCP non attraversano il confine L3.
```

2.2.6 Controllo d'accesso (Access Control)

2.2.6.1 Obiettivo. (i) autenticazione all'ingresso, (ii) autorizzazione fine a cosa può fare l'endpoint, (iii) enforcement sullo switch con ACL/Port Security.

2.2.6.2 802.1X (port-based NAC). **Supplicant** (host), **Authenticator** (switch) e **Authentication Server** (tipicamente RADIUS) negoziano l'accesso usando **EAP** (extensible authentication protocol) veicolato su **EAPOL** (EAP over LAN). 802.1X supporta credenziali diverse (username/password o certificati) e lega l'identità alla *porta* di switch all'inizio della sessione; il controllo si appoggia a RADIUS (EAP relay/termination) e ai tipi di messaggi EAPOL (Start, EAP, Key, Logoff). In tal modo lo switch può ammettere/negare la porta e associare policy all'identità autenticata.

Effetti attesi e limiti. Gli switch *802.1X-capable* mitigano *MAC spoofing* e *flooding* legando il MAC alla porta autenticata; restano però possibili attacchi fuori dal perimetro di 802.1X (es. *ARP poisoning*) e il *piggybacking* inserendo un piccolo hub/switch tra host e porta autenticata. L'autenticazione tra switch può creare un “inner core” fidato per impedire che un host si finga switch.

2.2.6.3 Autorizzazione dinamica con 802.1X. Dopo l'*auth*, il server può spingere parametri di **autorizzazione**:

- **VLAN per-utente** (membership assegnata dall'identità);
- **ACL per-utente** (policy di filtro associate all'identità);
- **UCL (User Control List)**: gruppi di utenti che condividono una stessa policy/ACL.

Queste informazioni sono veicolate nell'esito dell'autenticazione e consumate dallo switch per configurare forwarding e filtri coerenti con l'identità.

2.2.6.4 Port Security & ACL sugli switch. **Port Security** limita il numero (e optionalmente l'identità) dei MAC appresi per porta: blocca il *MAC flooding* e rende più difficile l'espansione non autorizzata della LAN aggiungendo switch clandestini. Le **ACL** a livello Ethernet non sono parte dello standard, ma gli switch moderni possono filtrare su MAC sorgente/destinazione o **Ethertype**; gli switch L3 estendono il match a campi L3/L4. In laboratorio, le ACL L2 si possono esprimere con **ebtables** (NETFILTER).

Listing 5: Esempio minimale di binding MAC→porta con ebtables

```
ebtables -A FORWARD --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP  
ebtables -A FORWARD --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP  
ebtables -A FORWARD --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP
```

(Le ACL del laboratorio usano **ebtables**/NETFILTER; le regole si applicano per catena e per livello.)

2.2.6.5 Segmentazione con VLAN (complemento). Le **VLAN 802.1Q** riducono il dominio di broadcast e separano il traffico L2; l'efficacia dipende dalla corretta configurazione (trunk, VLAN nativa, porte *access*). È pratica comune associare la VLAN all'identità 802.1X (VLAN dinamiche) e poi controllare il traffico inter-VLAN a L3. *Nota: i default non sono sicuri; configurazioni errate possono abilitare VLAN hopping.*

Laboratorio

2.3 Laboratorio 2A: ACL L2 con Linux Bridge ed `ebttables` (binding MAC→porta)

Questo laboratorio mostra come applicare un controllo d'accesso a Livello 2 vincolando, per ogni porta, i soli indirizzi MAC ammessi ($MAC \rightarrow porta$). L'obiettivo è accettare frame in ingresso solo dagli host attesi e bloccare tentativi di *MAC flooding/spoofing*.

Topologia di rete

Un host Linux con tre interfacce (`eth0`, `eth1`, `eth2`) funge da *bridge L2* (“switch” emulato). Su ciascuna porta è previsto un client:

- **client-1** collegato a `eth0`, MAC previsto `a0:a0:a0:a0:a0:a0`;
- **client-2** collegato a `eth1`, MAC previsto `b0:b0:b0:b0:b0:b0`;
- **client-3** collegato a `eth2`, MAC `c0:c0:c0:c0:c0:c0` da *non* accettare.

Goal: accettare solo i MAC previsti su ciascuna porta; verificare che il traffico in ingresso da **client-3** sia scartato.

Configurazione di esempio

2.3.0.1 1) Preparazione host (MAC/IP dei client). Esempio di impostazione (lato client) dei MAC e degli IP:

Listing 6: Impostazione MAC/IP sui client (esempio)

```
# client-1
ip link set dev eth0 address a0:a0:a0:a0:a0:a0
ip addr add 10.0.0.1/24 dev eth0

# client-2
ip link set dev eth0 address b0:b0:b0:b0:b0:b0
ip addr add 10.0.0.2/24 dev eth0

# client-3 (non ammesso)
ip link set dev eth0 address c0:c0:c0:c0:c0:c0
ip addr add 10.0.0.3/24 dev eth0
```

(Questi valori riflettono quelli delle slide del laboratorio.)

2.3.0.2 2) Creazione del bridge L2 (lato “switch” Linux). .

Listing 7: Bridge L2 con tre porte

```
# crea il bridge e collega le porte
ip link add name bridge type bridge
ip link set bridge up
ip link set dev eth0 master bridge
ip link set dev eth1 master bridge
ip link set dev eth2 master bridge
```

Tutte le porte sono ora nello stesso dominio L2; senza ACL, i tre client comunicano liberamente.

2.3.0.3 3) ACL L2 con ebttables (NETFILTER).

Listing 8: Binding MAC→porta con ebtables

```
# accetta su eth0 solo il MAC di client-1
ebtables -A FORWARD --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP
# accetta su eth1 solo il MAC di client-2
ebtables -A FORWARD --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP
# blocca su eth2 il MAC di client-3
ebtables -A FORWARD --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP

# opzionale: replica le stesse policy anche sulla chain INPUT del bridge-host
ebtables -A INPUT --in-interface eth0 -s ! a0:a0:a0:a0:a0:a0 -j DROP
ebtables -A INPUT --in-interface eth1 -s ! b0:b0:b0:b0:b0:b0 -j DROP
ebtables -A INPUT --in-interface eth2 -s c0:c0:c0:c0:c0:c0 -j DROP
```

Le ACL sono espresse nel framework **NETFILTER**: ebttables opera a L2 (MAC/EtherType), iptables/ip6tables a L3/L4.

Funzionamento del laboratorio

Le regole applicano un vincolo di identità L2: ogni porta accetta solo i frame dal MAC atteso; frame da MAC diversi vengono droppati (*anti-spoofing* di base, freno al *flooding*).

Verifica del comportamento

- **client-1 ↔ client-2**: ping riuscito.
- **client-3 → altri**: i pacchetti in ingresso sono scartati (nessuna risposta al ping).

Suggerimento: osserva i contatori `ebtables -L -Lc` mentre invii traffico di test.

Osservazione pratica

Questo schema è didattico ma realistico: i vendor implementano meccanismi simili (*Port Security*, ACL L2) sugli switch per contenere spoofing/flooding alla porta.

2.4 Laboratorio 2B: MACsec su Linux (confidenzialità, integrità e anti-replay a L2)

Questo laboratorio mostra come attivare **MACsec** (IEEE 802.1AE) tra due host Linux per proteggere il traffico L2 con integrità, optionalmente cifratura e protezione *anti-replay*.

Topologia di rete

Due host (**client1**, **client2**) collegati alla stessa LAN Ethernet. Ogni host crea un'interfaccia `macsec0` legata alla NIC fisica (`eth0`) e configura *Secure Channel* (SC) e *Security Association* (SA) con chiavi simmetriche.

Configurazione di esempio

2.4.0.1 Host A (client1).

Listing 9: MACsec lato client1

```
ip link add link eth0 macsec0 type macsec
```

```
# SA TX di client1 (chiave K1), SA RX verso il MAC di client2 (chiave K2)
ip macsec add macsec0 tx sa 0 pn 1 on key 01 09876543210987654321098765432109
ip macsec add macsec0 rx address b0:b0:b0:b0:b0:b0 port 1
ip macsec add macsec0 rx address b0:b0:b0:b0:b0:b0 port 1 sa 0 pn 1 on key 02
    12345678901234567890123456789012
ip link set macsec0 up
ip addr add 10.100.0.1/24 dev macsec0
```

2.4.0.2 Host B (client2) . .

Listing 10: MACsec lato client2

```
ip link add link eth0 macsec0 type macsec
# SA TX di client2 (chiave K2), SA RX verso il MAC di client1 (chiave K1)
ip macsec add macsec0 tx sa 0 pn 1 on key 02 12345678901234567890123456789012
ip macsec add macsec0 rx address a0:a0:a0:a0:a0:a0 port 1
ip macsec add macsec0 rx address a0:a0:a0:a0:a0:a0 port 1 sa 0 pn 1 on key 01
    09876543210987654321098765432109
ip link set macsec0 up
ip addr add 10.100.0.2/24 dev macsec0
```

2.4.0.3 Opzioni di sicurezza e test. .

Listing 11: Cifratura e anti-replay; test con ping

```
# Cifratura dei frame e protezione anti-replay (opzionali)
ip link set macsec0 type macsec encrypt on
ip link set macsec0 type macsec replay on

# Test
ping 10.100.0.2 # da client1 (verifica su Wireshark l'intestazione MACsec)
```

Con questa configurazione, MACsec fornisce integrità; attivando `encrypt on` aggiungi confidenzialità. La suite predefinita è *GCM-AES-128*; *GCM-AES-256* è disponibile in estensione. La gestione chiavi dinamica è demandata a 802.1X-2010 (fuori dallo scopo di questo lab).

Funzionamento del laboratorio

MACsec inserisce, tra MAC header e payload, un *Security Tag* e calcola un ICV (codice d'integrità); i frame risultano protetti contro manomissioni e, se abilitata, cifrati in transito.

Verifica del comportamento

- **Integrità only:** con `encrypt off` vedi i Layer-3 in chiaro ma i frame hanno tag/ICV MACsec.
- **Cifratura on:** abilita `encrypt on` e osserva in Wireshark che il payload non è leggibile.
- **Anti-replay:** abilita `replay on` e verifica che ritrasmissioni artificiose vengano scartate.

Suggerimento: cattura sul link fisico `eth0` per osservare l'incapsulamento MACsec.

Osservazione pratica

MACsec elimina *sniffing/MITM* sul filo (anche in caso di flooding per timeout CAM), ma non impedisce DoS né abusi da host già autorizzati: serve comunque *hardening* (ACL/DAI/Port Security) e monitoraggio.

2.4.1 Secure Address Resolution (IPv4/IPv6)

2.4.1.1 IPv4: dal piano di switching ai vincoli per-presa. **DHCP Snooping** costruisce una tabella (IP, MAC, porta, VLAN) osservando i messaggi DHCP leciti; questa base di conoscenza alimenta la **Dynamic ARP Inspection (DAI)** che blocca risposte ARP incoerenti con i binding noti. **IP Source Guard** applica il binding direttamente a L2, filtrando i frame con IP sorgente non atteso su quella porta. *Limite pratico:* lo *scope* è per-switch; un apparato vede solo i lease che attraversano le sue porte. Progettare i domini DHCP in modo che il traffico passi dove serve lo snooping.

2.4.1.2 IPv4: proposte crittografiche. **S-ARP** estende ARP con un campo di autenticazione e un'infrastruttura di gestione chiavi; in alternativa, si può “estendere” la protezione L2 fornita da **MACsec** fino all'endpoint e al multicast. Queste soluzioni sono robuste sul piano tecnico, ma più complesse da distribuire.

2.4.1.3 IPv6: NDP sicuro. In IPv6, la sicurezza della risoluzione passa da **SEND (Secure Neighbor Discovery, RFC 3971/6494)**: usa **CGA** (Cryptographically Generated Addresses) e opzioni firmate per autenticare i messaggi NDP, fornendo un meccanismo alternativo a IPsec, specifico per la discovery. Nella pratica operativa, resta poco adottato; in molti contesti si preferiscono controlli sullo switch (es. RA/DHCPv6 Guard) per mitigare *router/neighbor spoofing*.

2.4.2 Security Monitoring

2.4.2.1 Firewall & DPI. I firewall delimitano il traffico tra segmenti (caso avanzato di ACL) e, nei prodotti moderni, possono operare a più livelli con **Deep Packet Inspection** e ricostruzione di sessione applicativa. Nel dominio L2, il concetto di “Ethernet firewall” è assorbito da *ACL di switch* e dallo stack *NETFILTER* (ebtables/iptables); i firewall tradizionali gestiscono i livelli superiori.

2.4.2.2 IDS/IPS e accesso al traffico. **IDS/IPS** identificano attacchi tramite firme/anomalie e necessitano di vedere i pacchetti: o in *inline* (come un firewall) oppure tramite **port mirroring (SPAN)**, che copia il traffico di porte selezionate verso una porta di ascolto. I tap passivi su rame/fibra sono alternative poco rilevabili, ma richiedono accesso fisico.

2.4.2.3 Dove il monitoring aiuta davvero. Il mirroring/IDS rende *osservabili* attacchi tipici L2 (es. *MAC flooding, double-tagging*): molte firme sono facili da catturare; l'analisi DPI può correlare eventi di *poisoning* e *replay*. In parallelo, le *ACL* sugli switch limitano la superficie a L2.

2.4.2.4 Messaggio chiave (dalle slide). Il livello di sicurezza cresce con l'impegno amministrativo: una Ethernet *out-of-the-box* resta insicura (MAC flooding, ARP spoofing, STP attacks sono banali). Anche con ACL/802.1X, l'ARP spoofing resta possibile senza *DHCP Snooping + ARP Inspection*; MACsec risolve *sniffing/MiTM* ma non *DoS/analisi del traffico*. *Conclusione:* servono difese stratificate e configurazioni accurate.

3 NET_03 — Virtual LANs (VLAN)

3.0.1 Definizione e motivazione

Gli **switch Ethernet** tradizionali segmentano i domini di collisione ma non i domini di broadcast. Ciò significa che tutti gli host connessi a uno switch appartengono allo stesso dominio di broadcast e ricevono tutti i frame inviati in broadcast (come richieste ARP o DHCP).

Questa architettura è semplice ma poco scalabile: in reti di grandi dimensioni, il traffico broadcast può saturare la banda disponibile e ogni problema (come un loop o un attacco) può propagarsi a tutti gli host della rete.

Per risolvere questo limite si introduce il concetto di **Virtual LAN (VLAN)**, cioè la creazione di *più domini di broadcast logici* all'interno della stessa infrastruttura fisica. Ogni VLAN rappresenta una “rete virtuale” indipendente, isolata logicamente dalle altre pur condividendo gli stessi apparati.

3.0.2 Limiti delle reti fisicamente separate

Storicamente, la separazione dei domini di broadcast veniva realizzata con **sottoreti IP fisiche**, ciascuna connessa a un proprio switch e router. Tuttavia questo approccio presenta diversi limiti:

- necessità di cablaggi distinti e di apparati separati per ogni subnet anche se gli switch sono sullo stesso piano (fisico) come mostrato in Figura 11;
- difficoltà di riconfigurazione in caso di spostamento di host tra subnet diverse;
- costi di gestione e manutenzione elevati.

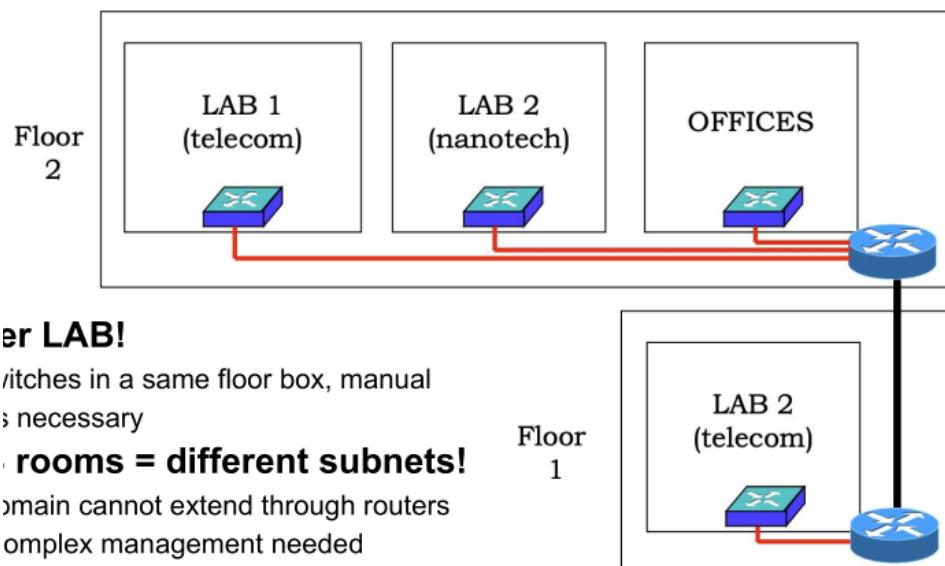


Figura 11: Physical ip subnet

Con l'introduzione degli **switch di Layer 3**, la velocità di routing non è più un problema, ma resta la necessità di una gestione logica più flessibile. Le VLAN forniscono questa flessibilità permettendo di isolare logicamente i gruppi di dispositivi in base a criteri funzionali, e non fisici.

3.0.2.1 Benefici principali delle VLAN

- **Confinamento del broadcast:** il traffico broadcast resta confinato all'interno della VLAN di appartenenza.
- **Scalabilità e ordine:** la rete può essere gestita come un insieme di domini separati, semplificando la diagnostica.
- **Sicurezza:** la separazione logica riduce la superficie d'attacco e impedisce la propagazione di minacce L2 tra gruppi diversi.

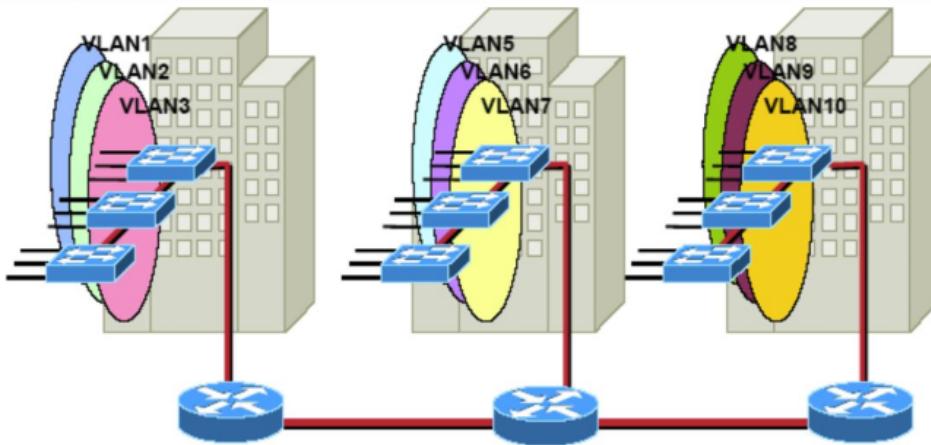


Figura 12: VLAN=area che limita il broadcast domain

3.1 Assegnazione e membership delle VLAN

3.1.1 Criteri di assegnazione

Un dispositivo può essere assegnato a una VLAN in base a diversi criteri:

- **Per porta (Port-based VLAN):** lo switch associa staticamente una VLAN a ogni porta. È il metodo più comune e lo standard definito da IEEE 802.1Q.
- **Per MAC address (User-based VLAN):** la VLAN è determinata dal MAC del dispositivo o dall'identità dell'utente autenticato (es. via 802.1X).
- **Per protocollo (Protocol-based VLAN):** introdotto da IEEE 802.1v, assegna la VLAN in base al protocollo di livello 3 (IP, IPX, ecc.).
- **Combinato (Cross-layer):** alcune implementazioni permettono regole gerarchiche, ad esempio prima per protocollo, poi per MAC, e infine per porta.

3.1.2 Vista logica

Ogni VLAN rappresenta un dominio di broadcast indipendente e, di conseguenza, è normalmente associata a una **sottorete IP dedicata**. La comunicazione tra VLAN diverse richiede un dispositivo L3 (router o Layer 3 switch (figura 13)) che svolga la funzione di *inter-VLAN routing*.

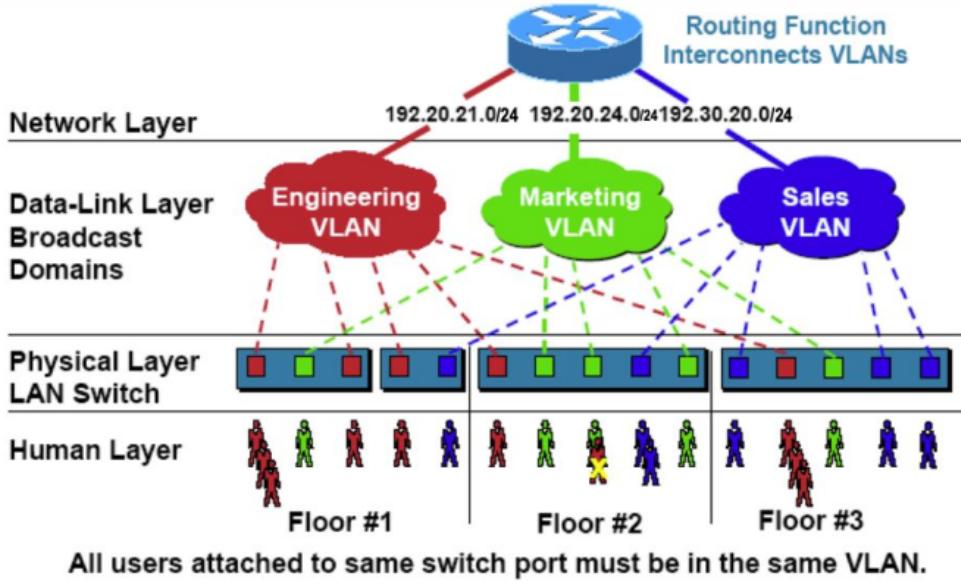


Figura 13: Physical vs logical view

3.1.2.1 Router “one-armed” In una configurazione one-armed router, una **singola interfaccia fisica** del router viene utilizzata per gestire **più VLAN** contemporaneamente. Ciò avviene tramite la creazione di **sub-interfacce virtuali**, ognuna associata a una VLAN specifica.

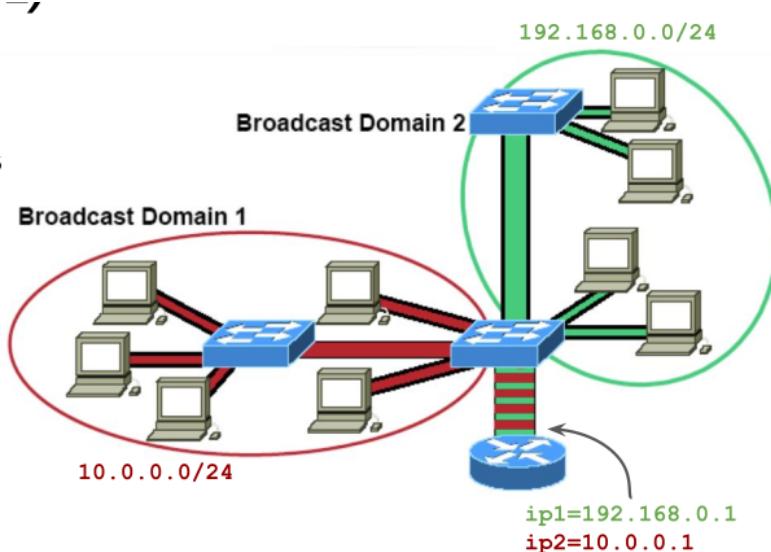


Figura 14: VLAN e sottoreti IP collegate tramite sub-interfacce

Listing 12: Esempio di configurazione router one-armed

```
ip link add link eth0 name eth0.10 type vlan id 10
ip link add link eth0 name eth0.20 type vlan id 20
ip addr add 10.0.10.1/24 dev eth0.10
ip addr add 10.0.20.1/24 dev eth0.20
```

In questo esempio, l’interfaccia fisica `eth0` viene suddivisa in due sub-interfacce virtuali, `eth0.10` e `eth0.20`, rispettivamente associate alle VLAN 10 e 20. A ciascuna sub-interfaccia viene assegnato un indirizzo IP appartenente alla sottorete della relativa VLAN.

In questo modo, il router può fornire servizi di **routing inter-VLAN**, agendo come **gateway predefinito** per ciascuna rete logica, pur utilizzando una sola porta fisica. Tale approccio è comune nei laboratori e negli ambienti di test, dove si vuole semplificare la topologia riducendo il numero di interfacce fisiche necessarie.

3.2 Trasporto dei frame: Tagging IEEE 802.1Q

Lo standard **IEEE 802.1Q** permette di far convivere più VLAN sullo stesso collegamento fisico. Per farlo, inserisce all’interno dei frame Ethernet un piccolo campo aggiuntivo, chiamato *tag VLAN*, che identifica la VLAN di appartenenza del frame. A seconda del tipo di collegamento, gli switch 802.1Q gestiscono questi tag in modo diverso, distinguendo tre tipologie di porte.

3.2.1 Porte di tipo Access, Trunk e Hybrid

- **Access Port:** collega un host finale (ad esempio un PC o una stampante). I frame che transitano su una porta di questo tipo sono sempre *senza tag (untagged)*. Lo switch associa internamente la porta a una specifica VLAN, aggiungendo o rimuovendo automaticamente il tag durante il transito interno.
- **Trunk Port:** collega due apparati di rete (ad esempio switch–switch o switch–router). Trasporta frame appartenenti a più VLAN contemporaneamente, inviandoli e ricevendoli *con tag 802.1Q*. In questo modo, ciascun frame mantiene l’informazione sulla VLAN di origine anche attraverso un link condiviso.
- **Hybrid Port:** gestisce sia frame *taggati* che *non taggati*. I frame non taggati vengono automaticamente associati alla **Native VLAN**, mentre quelli taggati mantengono il proprio VLAN ID. Questo tipo di porta è utile quando si collegano dispositivi che supportano il tagging VLAN insieme ad altri che non lo supportano.

3.2.2 Access links

Un **Access link** è un collegamento che parte da una **porta di tipo Access**, ossia una porta configurata per appartenere a una singola VLAN. Questo tipo di collegamento è utilizzato per connettere dispositivi finali (come PC, stampanti o server) oppure piccoli hub o switch non gestiti.

I frame che transitano su una porta di tipo Access sono sempre *non taggati (untagged)*: gli host collegati inviano e ricevono normali frame Ethernet, senza alcuna informazione sulla VLAN. È lo switch che, internamente, associa la porta a una VLAN e gestisce l’inserimento o la rimozione del *tag 802.1Q* quando il frame entra o esce dalla rete VLAN-aware.

In questo modo, i dispositivi connessi non devono essere consapevoli dell’esistenza delle VLAN: dal loro punto di vista fanno parte semplicemente di una rete Ethernet dedicata, tipicamente corrispondente a una specifica sottorete IP.

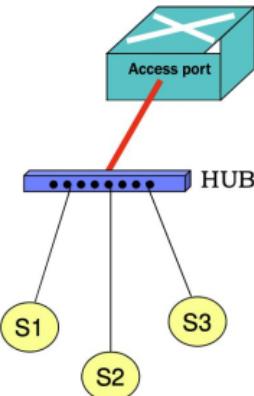


Figura 15: Esempio di collegamento Access link tra host e switch VLAN-aware

3.2.3 Access links nelle regioni legacy

In alcuni contesti, detti **legacy regions**, un access link può estendersi attraverso piccole LAN composte da più switch che non supportano le VLAN (*VLAN-unaware switches*). In questi casi, l'intera rete tradizionale viene vista dallo switch VLAN-aware come un unico segmento Ethernet appartenente a una sola VLAN.

Tutti i dispositivi collegati all'interno di tale regione condividono la stessa VLAN, anche se gli switch intermedi non gestiscono i tag 802.1Q. Questo approccio consente di integrare reti esistenti non VLAN-aware in un'infrastruttura moderna basata su VLAN.

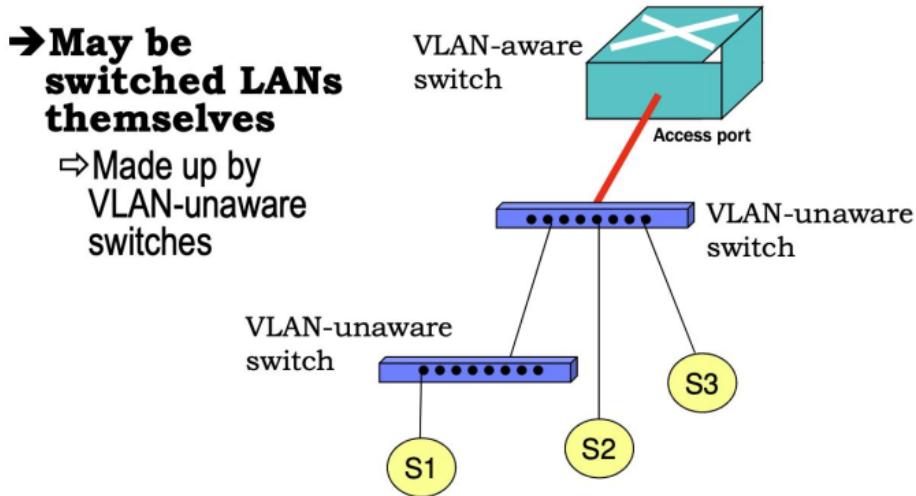


Figura 16: Access link esteso in una regione legacy con switch non VLAN-aware

3.2.4 Trunk links

Un **Trunk link** è un collegamento che parte da una **porta di tipo Trunk**, utilizzato per trasportare frame appartenenti a più VLAN contemporaneamente. È tipicamente impiegato nei collegamenti *switch–switch* o *switch–router*, dove è necessario far transitare traffico di più reti logiche sullo stesso mezzo fisico.

A differenza delle porte Access, le porte Trunk trasmettono e ricevono **frame taggati** con l'identificatore VLAN secondo lo standard **IEEE 802.1Q**. Il tag 802.1Q consente di distinguere

a quale VLAN appartiene ciascun frame, evitando la confusione tra traffici di reti diverse che condividono il link.

Un trunk link **non appartiene direttamente a una VLAN**, ma può trasportare:

- frame provenienti da *tutte* le VLAN configurate sullo switch;
- oppure frame appartenenti solo a un sottoinsieme di VLAN selezionate.

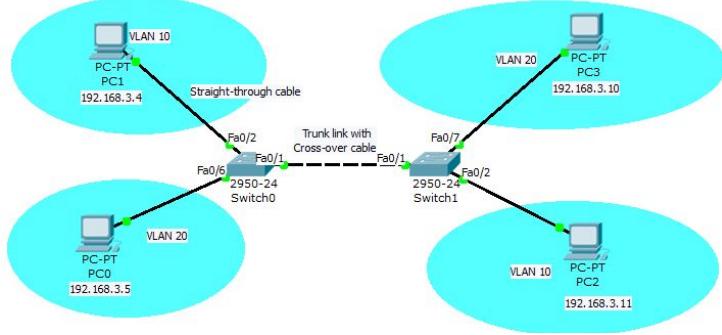


Figura 17: Esempio di collegamento Trunk tra apparati VLAN-aware

3.2.5 Hybrid links

Le **Hybrid links** rappresentano un'evoluzione dei trunk link e supportano sia **frame taggati** sia **frame non taggati**. I frame taggati mantengono il proprio VLAN ID, mentre i frame non taggati vengono associati a una VLAN predefinita (la **Native VLAN**).

Questo tipo di collegamento è utile quando sullo stesso link devono transitare:

- traffici di apparati VLAN-aware (che utilizzano frame taggati);
- e traffici di dispositivi legacy o VLAN-unaware (che inviano frame non taggati).

In sostanza, un hybrid link permette di far convivere traffico VLAN multiplo e traffico Ethernet standard sullo stesso collegamento, garantendo compatibilità tra apparati di diversa generazione. Nelle implementazioni moderne, molti switch trattano di fatto tutti i link come *ibridi*, in grado di gestire dinamicamente entrambe le tipologie di frame.

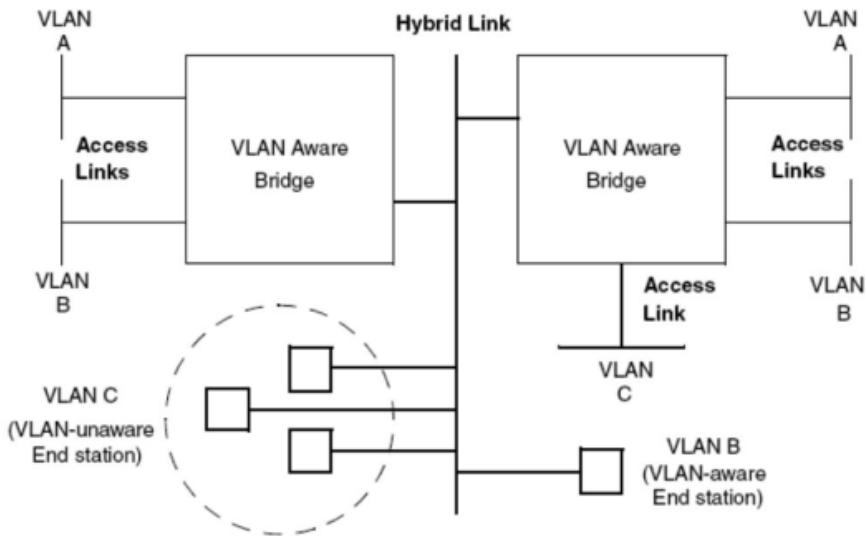


Figura 18: Esempio di Hybrid link che trasporta frame taggati e non taggati

In figura si nota come il collegamento ibrido consenta il transito simultaneo di traffico proveniente da VLAN diverse, includendo anche host non VLAN-aware (VLAN C), senza compromettere la separazione logica delle altre VLAN taggate.

3.2.6 Una stazione può appartenere a più VLAN?

In generale, un host connesso tramite una **Access port** appartiene a una sola VLAN, poiché i frame che transitano su quella porta sono sempre *non taggati* e associati a una singola rete logica.

Tuttavia, è possibile che una stessa stazione appartenga a **più VLAN** contemporaneamente. Questo avviene quando la stazione dispone di una **interfaccia trunk**, in grado di inviare e ricevere *frame taggati 802.1Q* appartenenti a VLAN diverse.

Un caso tipico è quello dei **server multi-VLAN**, che devono comunicare con più reti logiche (o sottoreti IP) attraverso un'unica interfaccia fisica. In tali scenari, il sistema operativo del server crea **sub-interfacce virtuali** (es. `eth0.10`, `eth0.20`), ciascuna configurata con un VLAN ID differente, consentendo la separazione logica del traffico pur utilizzando la stessa scheda di rete.

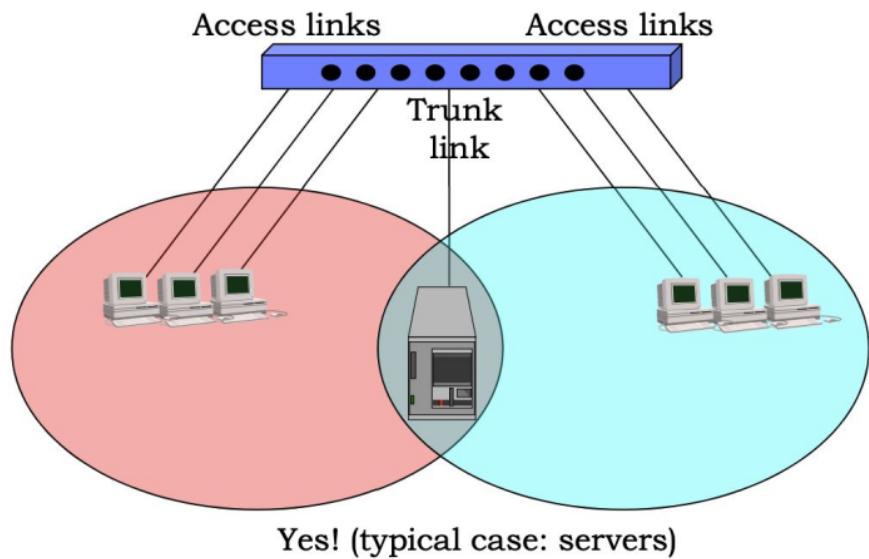


Figura 19: Esempio di stazione connessa a più VLAN tramite interfaccia trunk

In sintesi, solo le stazioni dotate di interfacce *VLAN-aware* possono appartenere a più VLAN: sono esse a gestire il tagging e l'instradamento del traffico tra le diverse reti logiche.

Laboratorio

3.3 Laboratorio 3: Configurazione VLAN e router one-armed

In questo laboratorio si analizza il funzionamento delle VLAN e del **routing inter-VLAN** attraverso un router connesso tramite una singola interfaccia fisica (*one-armed router*). L'obiettivo è comprendere come i frame vengano trasportati attraverso collegamenti di tipo *Access*, *Trunk* e *Hybrid* secondo lo standard IEEE 802.1Q.

Topologia di rete

La rete è composta da:

- uno **switch VLAN-aware** con tre porte configurate come Access (VLAN 10 e 20) e una porta configurata come Trunk verso il router;
- due host collegati alle porte Access, ciascuno appartenente a una VLAN distinta;
- un router configurato con sub-interfacce virtuali (eth0.10, eth0.20) per fornire connettività inter-VLAN.

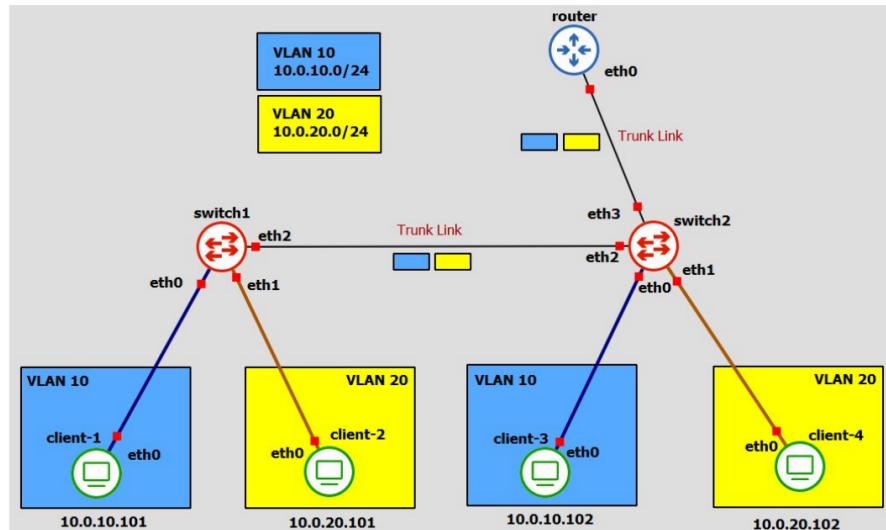


Figura 20: Topologia del Laboratorio 3: VLAN e router one-armed

Configurazione di esempio

Listing 13: Configurazione delle VLAN sul router one-armed

```
# Creazione delle sub-interfacce VLAN
ip link add link eth0 name eth0.10 type vlan id 10
ip link add link eth0 name eth0.20 type vlan id 20

# Assegnazione degli indirizzi IP (gateway delle rispettive VLAN)
ip addr add 10.0.10.1/24 dev eth0.10
ip addr add 10.0.20.1/24 dev eth0.20

# Attivazione delle interfacce
ip link set eth0.10 up
ip link set eth0.20 up
```

Sul lato switch:

Listing 14: Configurazione delle VLAN sullo switch

```
# Creazione VLAN
vlan 10
vlan 20

# Assegnazione delle porte
interface eth1
switchport mode access
switchport access vlan 10

interface eth2
switchport mode access
switchport access vlan 20

# Porta trunk verso il router
interface eth0
switchport mode trunk
switchport trunk allowed vlan 10,20
```

Funzionamento del laboratorio

Il laboratorio ha lo scopo di mostrare in modo pratico il funzionamento delle **VLAN** e del **router one-armed**, ovvero una configurazione in cui un unico collegamento fisico tra router e switch trasporta il traffico di più VLAN tramite **frame taggati IEEE 802.1Q**.

3.3.0.1 Isolamento del traffico Gli host collegati alle **porte Access** appartengono ciascuno a una VLAN distinta. I frame che transitano su queste porte sono *non taggati* e vengono separati dallo switch in base alla VLAN di appartenenza. In questo modo, gli host di VLAN diverse non possono comunicare direttamente: lo switch isola i domini di broadcast e impedisce la comunicazione a livello 2.

3.3.0.2 Routing inter-VLAN Per permettere la comunicazione tra VLAN diverse, il traffico deve passare attraverso il router. La connessione tra router e switch avviene tramite una **porta di tipo Trunk**, che trasporta i frame di più VLAN aggiungendo un tag 802.1Q a ciascun frame. Sul router, l'interfaccia fisica (ad esempio `eth0`) è suddivisa in più **sub-interfacce virtuali** (`eth0.10`, `eth0.20`, ecc.), ognuna configurata con:

- un **VLAN ID** specifico;
- un indirizzo IP che funge da **gateway** per la relativa VLAN.

Quando un host della VLAN 10 invia un pacchetto verso un host della VLAN 20:

1. il frame raggiunge lo switch sulla porta Access e viene inoltrato sul trunk verso il router con tag VLAN 10;
2. il router riceve il frame su `eth0.10`, lo elabora a livello 3 e decide di inoltrarlo sulla sub-interfaccia `eth0.20`;
3. il router rimanda il frame allo switch, questa volta con tag VLAN 20;
4. lo switch rimuove il tag e lo invia alla porta Access corrispondente alla VLAN 20.

Communicating between VLANs? Only via R1!!!

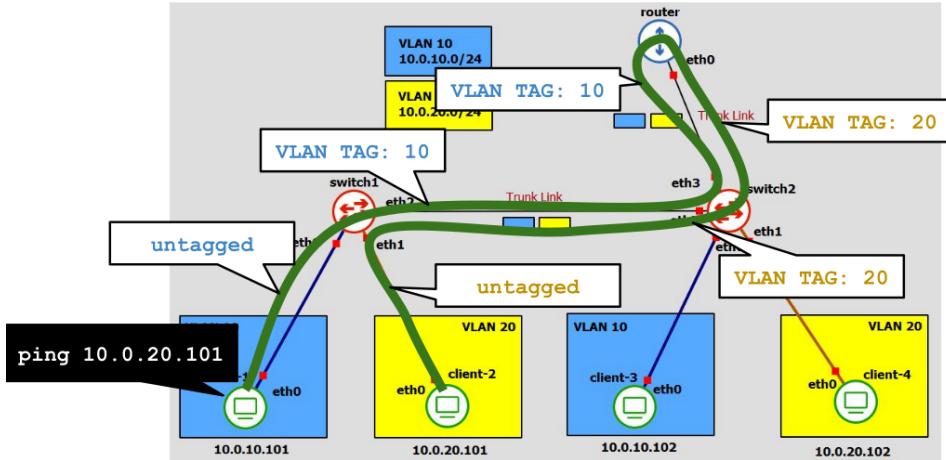


Figura 21: Comunicazione tra diverse vlan

3.3.0.3 Verifica del comportamento Nel laboratorio si può verificare che:

- gli host della stessa VLAN comunicano direttamente a livello 2, senza passare dal router;
- la comunicazione tra VLAN diverse avviene tramite il router (routing inter-VLAN);
- tutto il traffico tra router e switch è trasportato sul link trunk mediante frame taggati 802.1Q.

3.3.0.4 Osservazione pratica Catturando il traffico con `tcpdump` o `Wireshark` sull'interfaccia trunk, è possibile osservare i **tag VLAN** all'interno dei frame Ethernet. Ciò consente di verificare visivamente il meccanismo di separazione e instradamento del traffico tra le diverse VLAN.

3.4 Sicurezza delle VLAN e vulnerabilità di livello 2

Le VLAN migliorano l'isolamento logico del traffico, ma non eliminano le minacce presenti a livello di collegamento. Un attaccante connesso alla rete locale può sfruttare debolezze dei protocolli di livello 2 (Ethernet e ARP) o configurazioni errate degli switch per intercettare, modificare o dirottare il traffico di rete.

3.4.1 Minacce principali

3.4.1.1 1) MAC Flooding (CAM Overflow) Gli switch mantengono in memoria una **Content Addressable Memory (CAM)**, che associa indirizzi MAC a porte fisiche. Un attaccante può inviare migliaia di frame con indirizzi MAC falsi, riempiendo la tabella CAM e provocando un *overflow*. Quando la tabella è satura, lo switch non riesce più a determinare su quale porta si trova un determinato MAC e inizia a inoltrare i frame in **broadcast**, esponendo il traffico all'attaccante.

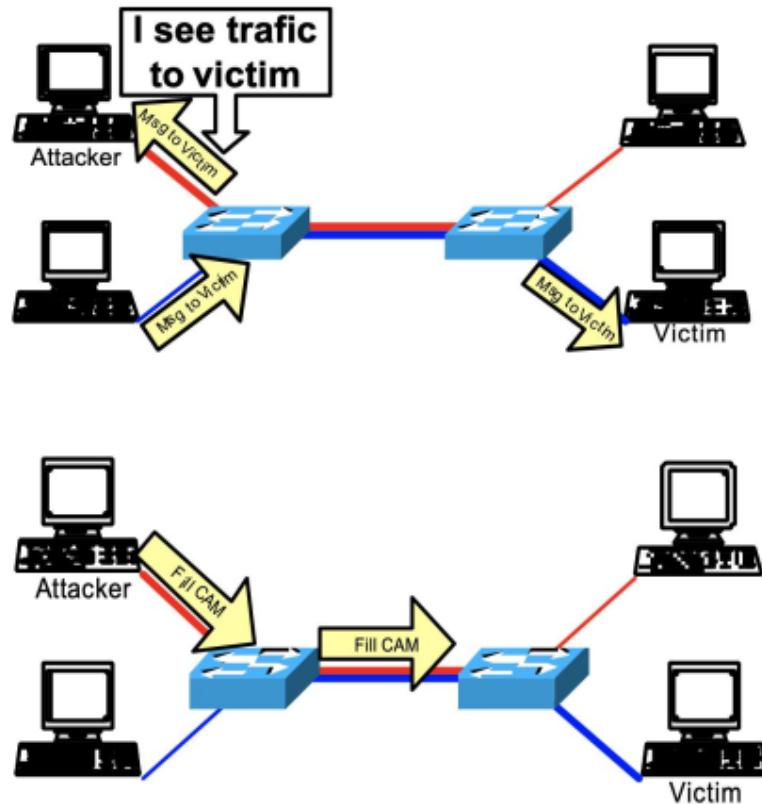


Figura 22: Esempio di MAC flooding

Mitigazione: abilitare la **port security**, limitando il numero di MAC address appresi per ciascuna porta, e disattivare l'apprendimento dinamico dove non necessario.

3.4.1.2 2) ARP Spoofing / Poisoning Il protocollo ARP non prevede autenticazione, quindi un attaccante può inviare **risposte ARP falsificate** per associare il proprio MAC all'indirizzo IP del gateway o di altri host nella stessa VLAN. In questo modo, intercetta o altera il traffico tra due dispositivi (*Man-in-the-Middle*).

Mitigazione: configurare **ARP statici** per i dispositivi critici, utilizzare strumenti di monitoraggio come `arpwatch` o implementare sistemi di protezione come **Dynamic ARP Inspection (DAI)** sugli switch gestiti.

3.4.1.3 3) VLAN Hopping Questa categoria di attacchi consente a un host di inviare o ricevere traffico appartenente a una VLAN diversa da quella assegnata, violando l'isolamento logico. Le due tecniche principali sono:

- **Basic VLAN hopping:** l'attaccante sfrutta il protocollo DTP (*Dynamic Trunking Protocol*) per negoziare automaticamente una connessione di tipo trunk con lo switch, ottenendo accesso a più VLAN.

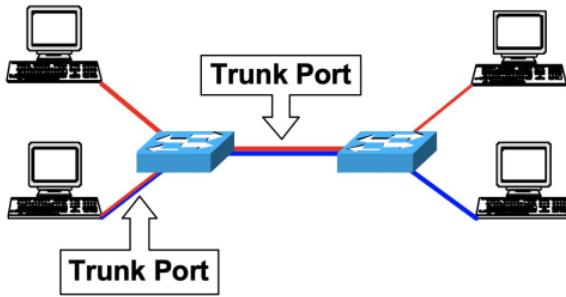


Figura 23: Collegamento tra due switch tramite porte trunk che trasportano il traffico di più VLAN sullo stesso link fisico.

- **Double Tagging:** il frame viene costruito con due tag 802.1Q annidati. Il primo tag (relativo alla VLAN nativa) viene rimosso dallo switch di ingresso, lasciando il secondo, che identifica la VLAN bersaglio.

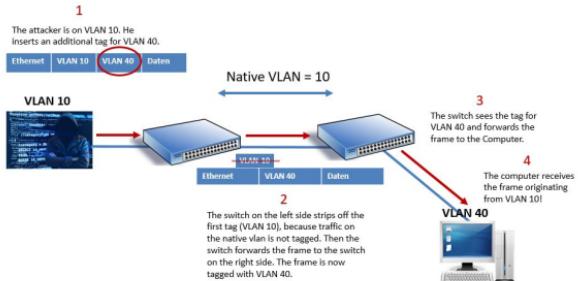


Figura 24: Esempio di attacco **VLAN Double Tagging**: un host malevolo appartenente alla VLAN 10 inserisce due tag 802.1Q (VLAN 10 e VLAN 40). Il primo switch rimuove il tag della VLAN nativa (10) e inoltra il frame, che conserva il secondo tag (40). Il frame attraversa quindi il trunk ed entra nella VLAN 40, violando l'isolamento tra VLAN.

Mitigazione: disabilitare DTP e configurare manualmente le porte come **Access** dove necessario; evitare l'uso della **VLAN 1** come VLAN nativa e assegnare VLAN native dedicate non utilizzate per il traffico utente.

3.4.1.4 4) Attacchi ai protocolli di controllo Oltre agli attacchi diretti alle VLAN, anche i protocolli di **controllo e gestione** utilizzati dagli switch di livello 2 possono essere sfruttati

da un attaccante per alterare la topologia della rete o raccogliere informazioni sensibili. Tra i principali:

3.4.1.4.1 Spanning Tree Attack (BPDU spoofing)

- **Cosa fa lo STP:** gli switch si scambiano BPDU (bridge protocol data units) per eleggere il *root bridge* e stabilire quali porte siano forwarding o blocking, evitando loop.
- **Come attacca l'avversario:** l'attaccante invia BPDU falsi con una priorità molto bassa (o con un bridge ID fittizio), facendo credere agli switch che il suo dispositivo sia il nuovo root.
- **Effetto pratico:** la topologia si ricalcola; alcuni link possono essere forzati in forwarding creando percorsi non previsti, perdita di connettività o instradamento del traffico attraverso il nodo dell'attaccante.
- **Mitigazione:** *BPDU Guard* spegne la porta se riceve BPDU su una porta che dovrebbe essere una porta access (cioè verso host), mentre *Root Guard* impedisce a un certo segmento di diventare root forzando il comportamento previsto.

3.4.1.4.2 VTP Attack (VLAN Trunking Protocol)

- **Cosa fa VTP:** permette di distribuire automaticamente la lista delle VLAN a tutti gli switch del dominio VTP.
- **Come attacca l'avversario:** un dispositivo malevolo si presenta come **server VTP** con una *revision number* superiore; gli altri switch accettano la nuova configurazione e sovrascrivono le VLAN locali.
- **Effetto pratico:** le VLAN possono essere cancellate o modificate su larga scala, causando indisponibilità o perdita dell'isolamento tra reti.
- **Mitigazione:** impostando VTP in *transparent* o disabilitandolo si evita che uno switch accetti e propaghi automaticamente configurazioni provenienti da fonti non affidabili.

3.4.1.4.3 Cisco Discovery Protocol Attack (information leakage)

- **Cosa fa CDP:** i dispositivi Cisco pubblicano informazioni (IP, modelli, versioni) su CDP verso i vicini.
- **Come attacca l'avversario:** un host malintenzionato cattura i pacchetti CDP o ascolta il traffico e ottiene dettagli utili per attacchi mirati (es. versioni vulnerabili).
- **Effetto pratico:** ricognizione facilitata: l'attaccante conosce quali dispositivi e software colpire.
- **Mitigazione:** disabilitando CDP sulle porte utenti si riduce la quantità di informazioni esposte ai terminali non fidati.

Laboratorio

3.5 Laboratorio 4: VLAN Hopping e Double Tagging

3.5.1 Scenario

L'obiettivo è dimostrare un attacco di **double tagging 802.1Q**: un host nella **VLAN nativa (VLAN 1)** tenta di inviare traffico a una vittima in **VLAN 20** *senza* routing inter-VLAN, sfruttando la rimozione del tag della VLAN nativa sul primo switch.

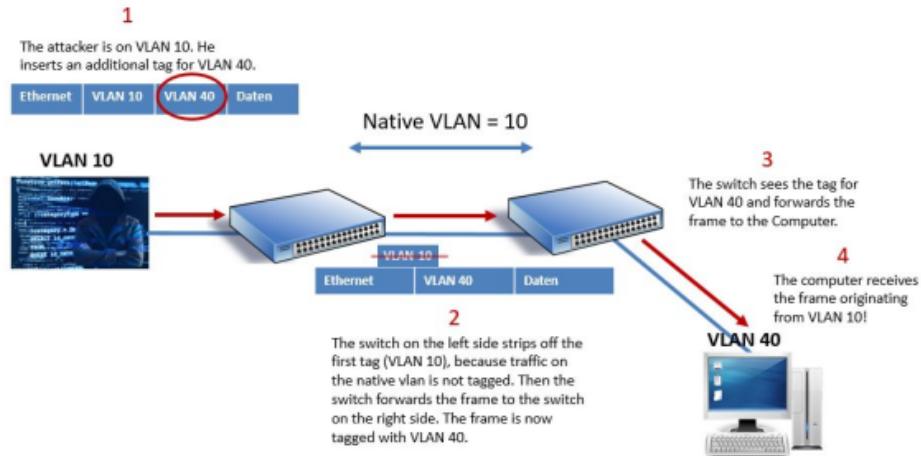


Figura 25: Attacco Double Tagging su trunk con VLAN nativa impostata a 1.

3.5.2 Topologia del laboratorio

La topologia è in Figura 26: attaccante in **VLAN 1 (nativa)** connesso a una porta che insiste sul *trunk*; vittima in **VLAN 20** su uno switch a valle; nessun router tra le VLAN.

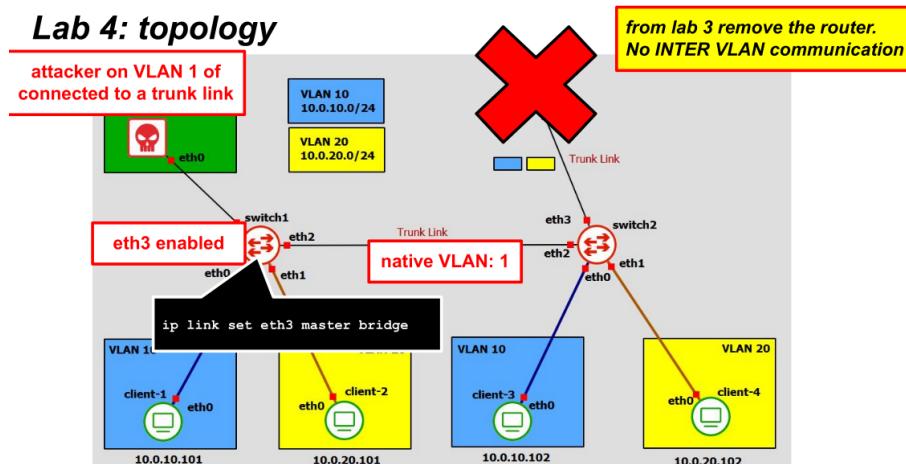


Figura 26: Lab 4 — Topologia: attaccante su VLAN 1 (nativa) verso trunk; vittima in VLAN 20; *no* inter-VLAN routing.

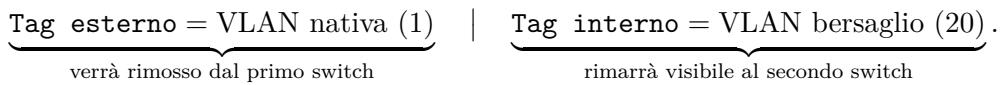
Nota operativa. Le porte `eth3` dei due switch formano il *trunk* con **VLAN nativa = 1**. L'attaccante è su una porta *access* in VLAN 1; gli host `client-3` (VLAN 10) e `client-4` (VLAN 20) sono su porte *access* degli switch opposti.

3.5.3 Prerequisiti perché l'attacco riesca

- esiste un **link trunk** tra due switch;
- stessa **VLAN nativa** configurata su entrambe le estremità del trunk (qui: VLAN 1);
- l'attaccante è connesso a **VLAN 1** (nativa) e può inviare frame con *doppio tag*;
- la vittima appartiene a una **VLAN diversa** (qui: VLAN 20).

3.5.4 Funzionamento dell'attacco (passo-passo)

L'attaccante costruisce un frame con **due tag 802.1Q annidati**:



1. **Invio dal nodo malevolo.** Il frame parte dall'attaccante con due tag: VLAN 1 (esterno) e VLAN 20 (interno).
2. **Primo switch (ingresso trunk).** Per definizione, i frame *taggati con la VLAN nativa* sul trunk vengono trattati come *untagged*: lo switch **rimuove il tag esterno** (VLAN 1) e inoltra il frame sul trunk *senza quel tag*.
3. **Attraversamento del trunk.** Il frame prosegue e, non avendo più il tag esterno, **rimane solo il tag interno** (VLAN 20) nel payload.
4. **Secondo switch (uscita trunk).** Vede un frame *ancora taggato* con **VLAN 20** (il tag interno) e lo inoltra correttamente all'*access port* della VLAN 20.
5. **Consegna alla vittima.** L'host in VLAN 20 riceve il frame come se provenisse da un nodo legittimo della propria VLAN.

Unidirezionalità. La risposta della vittima *non* segue lo stesso percorso: l'host in VLAN 20 non inserisce doppio tag e il frame di risposta non potrà rientrare verso l'attaccante in VLAN 1 senza routing. Per questo l'attacco è tipicamente **monodirezionale** (utile per inviare pacchetti/sondare servizi, meno per instaurare dialoghi completi).

3.5.5 Esecuzione pratica (Linux)

Listing 15: Costruzione di un frame con doppio tag via sub-interfacce annidate

```
ip link add link eth0 name eth0.1 type vlan id 1
ip link set eth0.1 up
ip link add link eth0.1 name eth0.1.20 type vlan id 20
ip link set eth0.1.20 up
ip addr add 10.0.20.250/24 dev eth0.1.20
arp -s 10.0.20.102 <MAC-vittima> -i eth0.1.20
ping 10.0.20.102
```

In questo modo lo `stack` inserisce **due tag 802.1Q** (outer=1, inner=20). Il `ping` raggiunge la vittima, ma la risposta generalmente non torna all'attaccante (vedi unidirezionalità).

3.5.5.1 Cosa osservare con lo sniffer

- sul **primo switch lato trunk**: si vedono frame che escono *senza* tag nativo e *con* il tag interno (20) ancora presente;
- sul **secondo switch**: si vedono frame **con VLAN 20** che vengono consegnati alla porta **access** della vittima;
- sull'host attaccante: assenza di risposte ICMP (o ARP) dalla vittima per la natura monodirezionale.

3.5.5.2 Limitazioni pratiche

- se la **VLAN nativa non è usata** (o è diversa e dedicata), il meccanismo di rimozione del primo tag non produce l'effetto desiderato;
- molti switch moderni applicano **filtr** su frame anomali (ad es. doppio tag con *outer* uguale alla nativa).

3.5.5.3 Mitigazioni

- **Disabilitare l'auto-trunking (DTP)** e configurare *manualmente* le porte utente come **access**;
- **Non usare VLAN 1 come nativa**; scegliere una **VLAN nativa dedicata** e non operativa per il traffico utente;
- **Consentire sul trunk solo le VLAN necessarie** (*allowed VLAN list*);
- abilitare controlli di **storm/unknown-unicast** e ispezioni su frame con *double tag* dove supportato;
- monitoraggio con IDS/port mirroring su link trunk per **pattern di doppio tagging**.

Conclusioni

Il **double tagging** sfrutta la gestione della **VLAN nativa** sui trunk per far accettare a uno switch un *tag interno* verso una VLAN diversa, *senza* routing. È spesso **monodirezionale**. La difesa passa da: hardening dei trunk (nativa dedicata, allowed-VLAN), porte utente in **access** statico, autenticazione 802.1X e controllo a livello di firewall/IDS.

4 NET_04 — 802.1x

4.1 Quadro generale e obiettivo

IEEE 802.1X è lo standard per il *Port-based Network Access Control* (PNAC): l'accesso alla rete avviene “per porta” a livello 2, così che solo i dispositivi autenticati possano oltrepassare l'autenticatore (switch/AP). Lavorando a L2, 802.1X evita processamento IP durante l'onboarding: i frame di autenticazione e i dati applicativi passano su interfacce logiche differenti, riducendo costi e superficie d'attacco. 802.1X definisce l'incapsulamento di EAP su LAN (**EAPoL**). EAP, invece, è un *framework* IETF indipendente dallo standard 802.1X.

4.2 Attori dell'architettura (vista di alto livello)

Tre ruoli: **Supplicant** (host/utente che richiede accesso), **Authenticator** (switch/AP che filtra e media), **Authentication Server** (AAA, tipicamente RADIUS). In stato iniziale la porta è *unauthorized* e lascia passare soltanto EAPoL; a successo la porta diventa *authorized* e il traffico dati è ammesso secondo la policy decisa dal server.

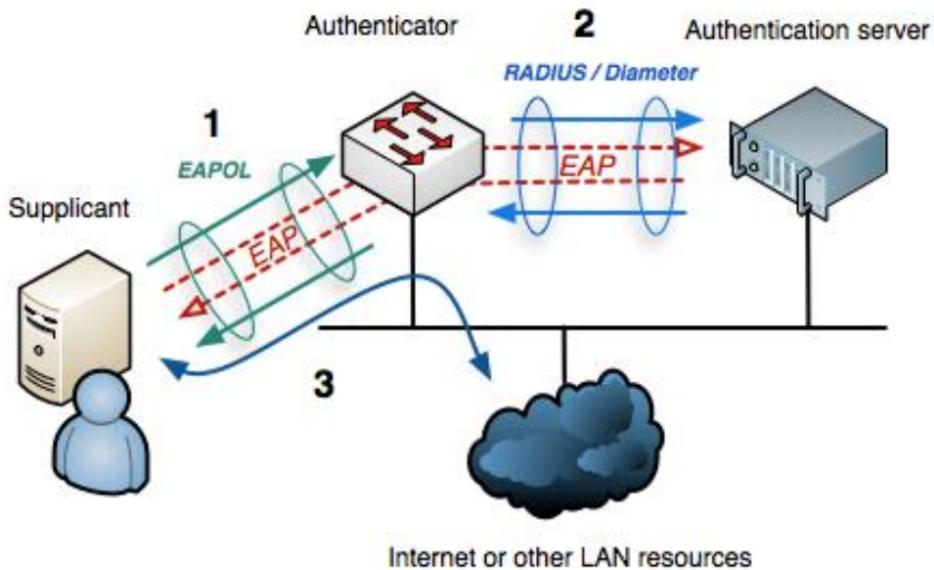


Figura 27: Architettura 802.1X e piani di traffico: (1) EAP incapsulato in EAPoL tra *supplicant* e *authenticator*; (2) EAP trasportato verso il server AAA tramite RADIUS/DIAMETER; (3) a autenticazione riuscita, sblocco del piano dati verso risorse LAN/Internet.

4.3 EAP ed EAPoL: fondamenti

4.3.0.1 EAP (Extensible Authentication Protocol). EAP è un *framework di autenticazione* standardizzato (RFC 3748, aggiornato da RFC 5247): definisce il *dialogo* tra peer (supplicant) e server e il formato dei messaggi, ma **non** impone un meccanismo crittografico unico. In altre parole, EAP stabilisce come negoziare ed eseguire un *metodo EAP* (la vera ricetta di autenticazione), mentre il trasporto su rete è demandato allo *strato sottostante*.

- **Che cos’è:** un livello di astrazione che fornisce messaggi *Request/Response* e *Success/Failure*, più un canale per scambiarsi i parametri del metodo scelto; esistono una quarantina di metodi (tra cui **EAP-TLS**, **PEAP**, **EAP-TTLS**, **EAP-SIM/AKA**, **EAP-FAST**).

- **Che cosa non è:** *non* è un *wire protocol* L2/L3; non cifra né autentica “da solo”; la sicurezza **dipende** dal metodo selezionato (es.: *EAP-MD5* non offre mutua autenticazione né protezione contro attacker attivi; *EAP-TLS* sì).
- **Dove gira:** EAP può essere incapsulato su vari “lower layer” (PPP, 802.11, 802.3); in 802.1X su Ethernet/Wi-Fi si usa **EAP over LAN (EAPoL)** per portare EAP tra supplicant e authenticator, che a sua volta inoltra verso AAA (tipicamente RADIUS).

Formato dei messaggi: un pacchetto EAP contiene **Code** (*Request, Response, Success, Failure*), **Identifier** (accoppia richiesta/risposta), **Length** e **Data** (il payload del metodo). Il server AAA decide l'esito del metodo e, se positivo, l'authenticator sblocca la porta e applica l'autorizzazione (VLAN/ACL).

4.4 EAP termination vs EAP relay (EAPoR)

Tra *authenticator* (switch/AP) e *server AAA* esistono due modelli operativi, con effetti concreti su sicurezza, scalabilità e troubleshooting.

4.4.0.1 EAP termination mode (terminazione locale). Lo switch *implementa* il metodo EAP e parla con il server usando i messaggi RADIUS tradizionali (*Access-Request/Accept/Reject*) senza trasportare il payload EAP end-to-end.

- **Dove “vive” il metodo:** dentro lo switch (NAS). Il server AAA verifica credenziali/attributi, ma non esegue direttamente il metodo EAP.
- **Vantaggi:** latenza ridotta; minor banda verso AAA; utile per metodi semplici (es. EAP-MD5 in lab) o quando si desidera *offload* di logica sul bordo.
- **Limiti:** meno flessibilità (aggiornare/estendere metodi richiede supporto sul NAS); minore *visibilità* diagnostica lato server; la derivazione delle chiavi (MSK) non arriva automaticamente al server, quindi integrazioni con altre fasi (es. roaming/derivazioni) sono più macchinose.
- **Uso tipico:** ambienti didattici o reti con metodo basico e policy semplici.

4.4.0.2 EAP relay mode (EAP over RADIUS, EAPoR). Lo switch *inoltra inalterati* i messaggi EAP del supplicant al server incapsulandoli negli attributi RADIUS **EAP-Message + Message-Authenticator**. Il metodo EAP viene eseguito *end-to-end* tra supplicant e server.

- **Dove “vive” il metodo:** sul *server AAA*. Lo switch resta un ponte L2/AAA “consapevole”.
- **Vantaggi:** massima compatibilità con metodi complessi (EAP-TLS, PEAP/TTLS, FAST, SIM/AKA); la **MSK/EMSK** è generata sul server e può alimentare altri servizi (es. derivazioni chiave, MACsec con dynamic CAK, Wi-Fi 802.11i); aggiornare/aggiungere metodi richiede solo aggiornare il server.
- **Sicurezza:** l'attributo **Message-Authenticator** protegge l'integrità del trasporto EAP su RADIUS; con link non fidati si usa RADIUS/TLS (RadSec) o IPsec per confidenzialità.
- **Diagnostica:** il server vede *tutto* lo scambio EAP (tracce/PCAP più utili); più semplice fare *troubleshooting* su handshake TLS, certificati, inner method, ecc.
- **Uso tipico:** quasi tutti i deployment reali 802.1X enterprise.

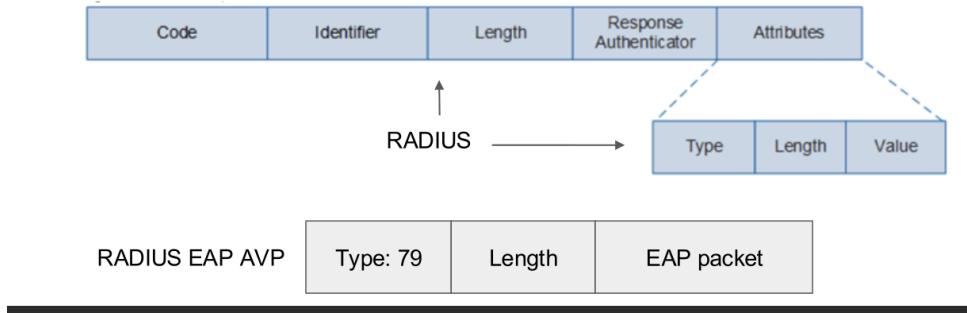


Figura 28: struttura pacchetto EAPoR

4.5 Processi di autenticazione: dai diagrammi alle implicazioni operative

4.5.0.1 Message exchange (visione di alto livello). Lo scambio coinvolge tre attori:

1. **Supplicant ↔ Authenticator (EAPoL):** il controllo d'accesso viaggia su *EAP over LAN*. Tipicamente: *EAP-Request/Identity* → *EAP-Response/Identity* e poi i messaggi del metodo EAP.
2. **Authenticator ↔ Server AAA (RADIUS/DIAMETER):** l'authenticator incapsula/-termina EAP e dialoga con il server usando messaggi *Access-Request/Challenge/Accept/Reject* (più accounting). In *relay* l'EAP del client è inoltrato end-to-end dentro l'attributo *EAP-Message*.
3. **Sblocco del piano dati:** a successo, l'authenticator cambia lo stato della porta in *authorized* e applica la policy (VLAN/ACL/UCL). In caso di fallimento o *EAPoL-Logoff*, la porta torna *unauthorized*.

Implicazioni: il troubleshooting si fa catturando EAPoL lato accesso (tcpdump ether proto 0x888e) e RADIUS lato control-plane; timer e ritrasmissioni sono gestiti dall'authenticator e impattano l'esperienza di login.

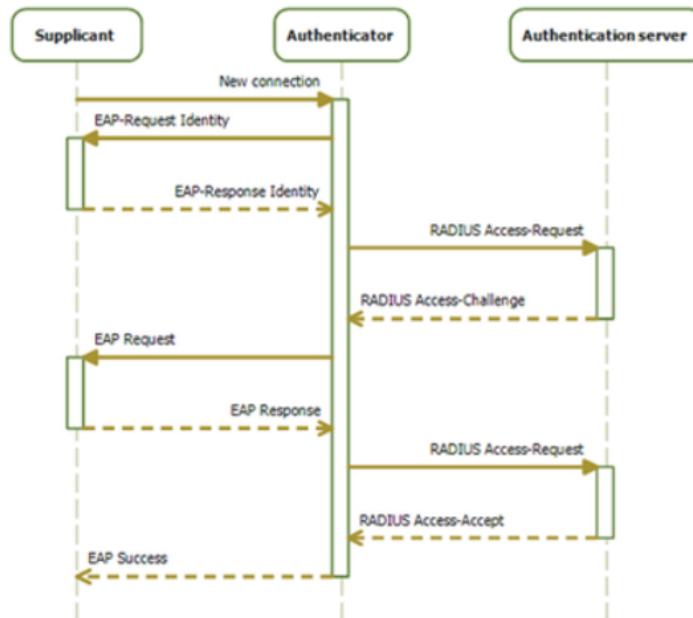


Figura 29: Message exchange ad alto livello

4.5.0.2 Relay Mode (EAP-MD5) — sequenza dettagliata. Nel *relay* l'authenticator non “capisce” il metodo: inoltra i messaggi EAP al server dentro RADIUS.

1. **EAPoL-Start** (client → authenticator): il supplicant innesca la procedura.
2. **EAP-Request/Identity** (authenticator → client): richiesta dell'identità.
3. **EAP-Response/Identity** (client → authenticator) ⇒ **Access-Request** con EAP-Message (authenticator → server).
4. **Access-Challenge** (server → authenticator): il server genera la *MD5 challenge*.
5. **EAP-Request/MD5-Challenge** (authenticator → client): inoltro della challenge.
6. **EAP-Response/MD5-Challenge** (client → authenticator): il client calcola $MD5(id + challenge + password)$ e risponde ⇒ **Access-Request** con EAP-Message (authenticator → server).
7. **Access-Accept/Reject** (server → authenticator): esito dell'autenticazione (in *Accept* possono esserci attributi di autorizzazione: VLAN, ACL, UCL).
8. **EAP-Success/Failure** (authenticator → client) e **cambio stato porta**: *authorized* a successo, *unauthorized* a fallimento.
9. **Handshake/keepalive**: l'authenticator interroga periodicamente il client (es. *EAP-Request/Identity*) per verificare che la sessione sia ancora viva; timeout consecutivi ⇒ chiusura sessione.
10. **EAPoL-Logoff** (client → authenticator): logout proattivo; l'authenticator ferma l'accouting, rimuove autorizzazioni e riporta la porta a *unauthorized*.

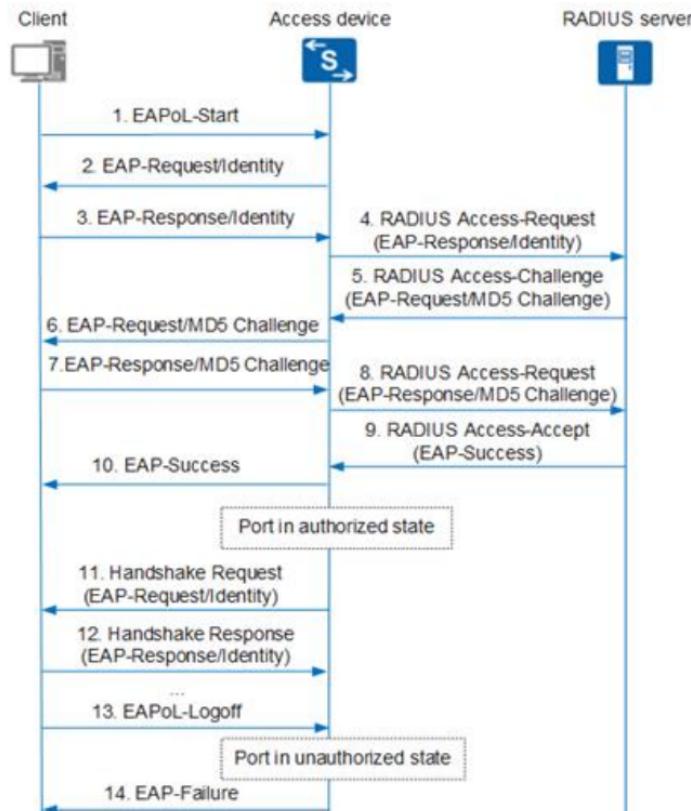


Figura 30: RelayMode md5 challenge

4.5.0.3 EAP-TLS: esempio più complesso. Rispetto a EAP-MD5, qui il processo è molto più sicuro perché si basa su un **handshake TLS** completo tra il client (supplicant) e il server AAA.

In pratica, dopo l'avvio (*EAPoL-Start*), l'autenticazione procede così:

- Il server invia una richiesta EAP di tipo **EAP-TLS**, che avvia la sessione TLS (come un normale *client_hello/server_hello* HTTPS).
- Il client risponde con il proprio **certificato** e completa la fase di *key exchange* e *cipher negotiation*.
- Entrambe le parti verificano i certificati: così si ottiene una **autenticazione reciproca (mutual authentication)**.
- Alla fine, viene derivata una **Master Session Key (MSK)** sicura, usata per proteggere il traffico o per altri protocolli (es. MACsec o WPA2-Enterprise).

Se qualcosa va storto (certificato non valido, handshake interrotto, errore TLS), il server invia *EAP-Failure*. In sintesi, EAP-TLS trasporta l'intero handshake TLS dentro EAP, fornendo un livello di sicurezza equivalente a una connessione HTTPS cifrata, ma applicato all'autenticazione di rete.

4.6 Operazioni aggiuntive: ri-autenticazione, logout, timer

Le slide enfatizzano aspetti gestionali spesso sottovalutati:

- **Re-auth:** se cambiano parametri (o stato) dell'utente, si forza ri-autenticazione; in condizioni anomale è previsto un flusso per utenti in *pre-connection* (ottimizza l'accesso “rapido”).
- **Logout e accounting:** se l'uscita non è rilevata (né dallo switch né da RADIUS), l'accounting resta “attivo”, creando incongruenze e possibili *spoof* su IP/MAC “appesi”. L'access device deve *immediatamente* rilevare logout, cancellare l'entry utente e fermare l'accounting.
- **Timer:** 802.1X si appoggia a timer per ritrasmissioni e timeout; la loro taratura impatta esperienza utente e affidabilità della sessione.

Tutti questi punti sono rappresentati esplicitamente nelle slide “Re-Authentication”, “Log out and Timers”.

4.7 Autorizzazione post-autenticazione: VLAN, ACL, UCL

4.7.0.1 VLAN dinamiche. Utenti non autenticati e risorse “ristrette” sono messi in VLAN diverse; a successo, il server assegna una *authorized VLAN*, che ha precedenza sulla configurazione statica dell'interfaccia. Attributi RADIUS standard obbligatori: *Tunnel-Type=VLAN(13)*, *Tunnel-Medium-Type=802(6)*, *Tunnel-Private-Group-ID=<VLAN>*. La VLAN statica torna attiva quando l'utente va offline.

4.7.0.2 ACL per-utente. Dopo l'autenticazione, il server può inviare all'*authenticator* una **Access Control List (ACL)** personalizzata per filtrare il traffico dell'utente. L'ACL definisce quali pacchetti possono attraversare la porta: i pacchetti che corrispondono a regole *permit* vengono inoltrati, mentre quelli che incontrano regole *deny* vengono bloccati.

Le ACL possono essere assegnate in due modi:

- **Assegnazione statica:** il server invia solo un riferimento all'ACL tramite l'attributo **Filter-Id**. Le regole vere e proprie devono essere già configurate localmente sull'apparato di accesso.
- **Assegnazione dinamica:** il server invia insieme al nome anche le *regole complete*, tramite attributi estesi RADIUS (ad esempio **HW-Data-Filter** nei sistemi Huawei). In questo modo le policy possono essere generate o aggiornate in tempo reale, senza modificare la configurazione dello switch.

Questa funzione consente di applicare controlli di traffico differenti per utente, ruolo o gruppo, rendendo la fase di *autorizzazione* molto più granulare e adattabile.

4.7.0.3 UCL (User Control List). La **User Control List (UCL)** è un meccanismo che permette di gestire in modo collettivo gruppi di utenti o dispositivi che condividono le stesse esigenze di accesso. Invece di assegnare policy individuali a ogni terminale, gli utenti vengono *raggruppati* in una UCL che eredita un insieme comune di regole.

Durante l'autenticazione, il server può:

- assegnare la UCL tramite il suo **nome**, usando l'attributo standard **Filter-Id**;
- oppure indicarla tramite un **identificatore numerico (ID)**, usando attributi RADIUS estesi, come **HW-UCL-Group** nei sistemi Huawei.

In entrambi i casi, la policy associata alla UCL deve essere *preconfigurata* sull'apparato di accesso, così che l'authenticator sappia quali permessi e restrizioni applicare agli utenti appartenenti a quel gruppo. Questo approccio semplifica notevolmente la gestione amministrativa, riducendo errori e duplicazioni nella configurazione delle policy di rete.

4.8 Vulnerabilità storiche e motivazione per MACsec

Se un attaccante intercetta i frame su una porta autorizzata (es. stesso hub del legittimo), può *spoofare* MAC/IP e accedere al mezzo; inoltre gli *EAPoL-Logoff* essendo in chiaro sono falsificabili per causare DoS. Per mitigare questi limiti la revisione 802.1X-2010 introduce **MACsec Key Agreement (MKA)** e, con MACsec, confidenzialità/integrità/anti-replay dei frame L2.

Laboratorio

4.9 Lab 5 — 802.1X Port-Based Authentication & VLAN assignment

4.9.0.1 Obiettivo. In questo laboratorio viene dimostrato il funzionamento del controllo d'accesso 802.1X su porta con autenticazione basata su **EAP-MD5** e l'assegnazione dinamica di VLAN da parte del server RADIUS. L'esperimento mostra come solo gli host autenticati possano generare traffico sulla rete e come l'autenticazione permetta al server di determinare la VLAN o le ACL applicate alla porta di accesso.

4.9.0.2 Topologia di rete. L'ambiente di test comprende:

- due host cablati che agiscono da *supplicant*;
- un server RADIUS (FreeRADIUS) che gestisce AAA e assegna VLAN dinamiche;
- uno switch Linux che funge da *authenticator* 802.1X e da router L3;
- un access point o gateway che fornisce l'accesso a Internet.

Le interfacce **eth0** e **eth1** dello switch rappresentano le porte di accesso controllate da 802.1X; **eth2** è la porta verso il server RADIUS; **eth3** collega la rete interna e fornisce connettività verso Internet.

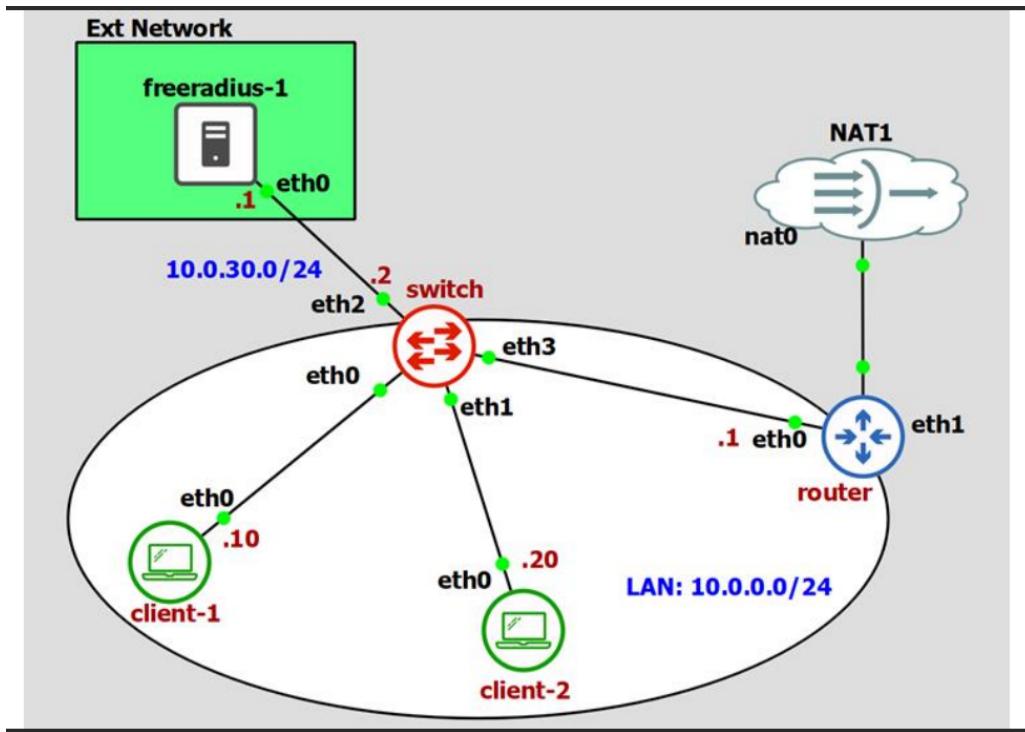


Figura 31: Topologia rete

4.9.0.3 Configurazione dello switch (Authenticator). Lo switch agisce da intermediario 802.1X grazie a `hostapd` in modalità `driver=wired`. Le interfacce di accesso sono inizialmente bloccate (*unauthorized*) e vengono sbloccate solo dopo l'autenticazione positiva. Le operazioni principali:

- creazione del `bridge` e associazione delle interfacce `eth0`, `eth1` e `eth3`;

- abilitazione dell'inoltro EAPoL con: `echo 8 > /sys/class/net/bridge/bridge/group_fwd_mask;`
- impostazione di policy DROP su `ebtables` per il traffico L2 non autorizzato;
- configurazione `hostapd.conf` con: `ieee8021x=1, use_pae_group_addr=1, auth_server_addr, auth_server_port, auth_server_shared_secret.`

4.9.0.4 Server di autenticazione (FreeRADIUS). Il server riceve le richieste RADIUS dagli *authenticator* e valida le credenziali degli utenti. La configurazione include:

- file `/etc/freeradius/3.0/clients.conf`: definisce i NAS autorizzati (es. lo switch Linux);
- file `/etc/freeradius/3.0/users`: definisce gli utenti e le VLAN da assegnare, ad esempio:

```
pippo Cleartext-Password := "pippo"
  Service-Type = Framed-User,
  Tunnel-Type = 13,
  Tunnel-Medium-Type = 6,
  Tunnel-Private-Group-ID = 10
```

```
pluto Cleartext-Password := "pluto"
  Service-Type = Framed-User,
  Tunnel-Type = 13,
  Tunnel-Medium-Type = 6,
  Tunnel-Private-Group-ID = 20
```

- avvio del demone con `freeradius -X` per visualizzare in tempo reale i pacchetti RADIUS.

4.9.0.5 Configurazione dei client (Supplicant). Ogni host utilizza `wpa_supplicant` in modalità wired con metodo **EAP-MD5**. Esempio di configurazione:

```
ap_scan=0
network={
  key_mgmt=IEEE8021X
  eap=MD5
  identity="pippo"
  password="pippo"
  eapol_flags=0
}
```

Gli host sono inizialmente configurati con IP statico per semplicità, ma in un contesto reale si può integrare un server DHCP che rilasci gli indirizzi dopo l'autenticazione.

4.9.0.6 Controllo del traffico L2. Per abilitare o bloccare dinamicamente le stazioni autorizzate, viene utilizzato uno **script Python** collegato a `hostapd_cli`. Lo script intercetta gli eventi *EAP-SUCCESS* ed *EAP-FAILURE* e modifica le regole di `ebtables`:

- aggiunge una regola ACCEPT per il MAC dell'utente autenticato;
- rimuove la regola quando l'utente si disconnette (**EAPoL-Logoff**) o fallisce la ri-autenticazione.

Le associazioni vengono salvate in un file JSON per garantire persistenza.

4.9.0.7 Sequenza operativa.

1. Il supplicant invia **EAPoL-Start**.

2. L'autenticator risponde con **EAP-Request/Identity**.
3. Il client replica con **EAP-Response/Identity**, incapsulato in **Access-Request** verso il server.
4. Il server RADIUS valuta le credenziali e restituisce **Access-Challenge** (MD5) o **Access-Accept**.
5. A successo, l'autenticator invia **EAP-Success** e abilita la porta: l'host entra nella VLAN autorizzata.
6. Se l'autenticazione fallisce, la porta resta bloccata e il traffico viene scartato.

4.9.0.8 Osservazioni e risultati.

- Gli host autenticati vengono inseriti automaticamente nella VLAN definita nel profilo RADIUS.
- Il controllo del traffico via `ebtables` garantisce isolamento tra utenti non autenticati e autenticati.
- I pacchetti EAPoL e RADIUS possono essere catturati con `tcpdump` per analizzare lo scambio.
- La struttura è facilmente estendibile ad altri metodi EAP (es. **EAP-TLS**) per autenticazione con certificati.

4.9.0.9 Conclusioni. Il laboratorio dimostra come lo standard IEEE 802.1X realizzi un **controllo d'accesso per porta** efficace e scalabile, spostando l'intelligenza di autenticazione sul server centrale e permettendo politiche per-utente (VLAN, ACL, UCL). La combinazione di `hostapd`, `FreeRADIUS` e `wpa_supplicant` rappresenta un'implementazione open-source completa del paradigma 802.1X.

4.10 MACsec e MKA (802.1AE + 802.1X-2010)

4.10.1 Perché MKA: il tassello mancante dopo 802.1X

Dopo l'autenticazione 802.1X, la porta diventa autorizzata ma il *traffico L2* rimane in chiaro e vulnerabile a spoofing e manomissione. **MACsec** (IEEE 802.1AE) aggiunge confidenzialità, integrità e anti-replay ai frame Ethernet sul *link locale*, mentre **MKA** (MACsec Key Agreement, parte di 802.1X-2010) *negozi e distribuisce* in modo sicuro le chiavi necessarie a MACsec. In breve: 802.1X decide *chi entra*, MACsec+MKA decide *come proteggere* i frame una volta dentro.

4.10.2 Le chiavi in gioco e come si ottengono

Il cuore di MKA è una piccola filiera crittografica.

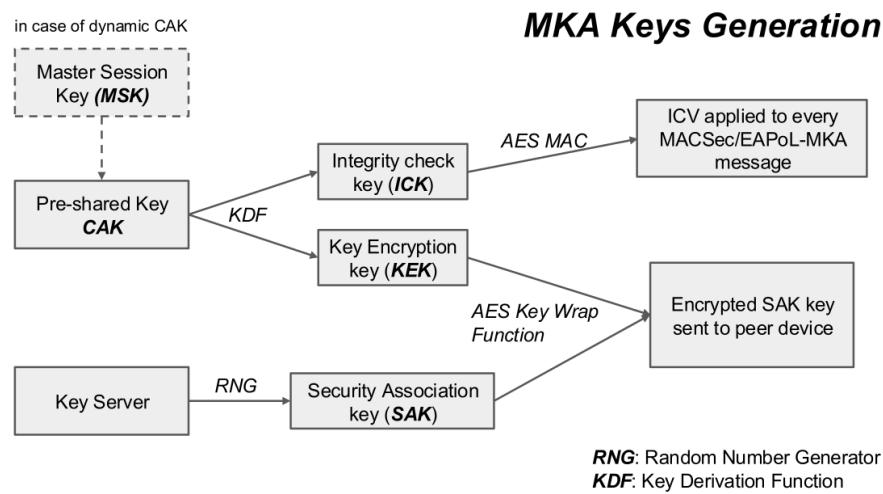


Figura 32: MKA keys generation: dal CAK (statico o derivato da MSK) si ottengono ICK e KEK (KDF); il Key Server genera la SAK (RNG) e la distribuisce cifrata (AES Key Wrap), mentre l'ICK protegge i messaggi EAPOL-MKA (ICV).

4.10.2.1 CAK e CKN La **CAK** (*Connectivity Association Key*) è la *chiave di base* condivisa (pre-shared) da tutti i partecipanti alla stessa *Connectivity Association* (CA), cioè al gruppo logico che userà MACsec su quel link. La CAK *non cifra i dati* direttamente: serve per proteggere il piano di controllo MKA e per derivare le chiavi operative. Il **CKN** (*Connectivity association Key Name*) è il *nome/identificatore della CA*: non è segreto, ma consente ai peer di riconoscere a quale associazione stanno aderendo e di far coesistere più CA sullo stesso mezzo. In pratica, *CKN individua il gruppo, CAK ne è il segreto condiviso*. Cambiando CKN o CAK si forza la (ri)formazione della CA e un nuovo ciclo di chiavi.

4.10.2.2 ICK e KEK A partire dalla CAK, MKA applica una **KDF** (Key Derivation Function) e ricava due chiavi di controllo con ruoli distinti: **ICK** (*Integrity Check Key*) e **KEK** (*Key Encryption Key*). L'ICK viene usata per calcolare un *ICV* (Integrity Check Value) sui messaggi EAPOL-MKA, garantendo che l'annuncio di capacità, l'elezione del Key Server e la distribuzione delle chiavi non possano essere manipolati. La KEK, invece, serve per *cifrare e incapsulare* (es. con AES Key Wrap) le chiavi dati che il Key Server invia ai peer. Nota importante: *ICK e KEK proteggono solo il piano di controllo*; il traffico utente su Ethernet verrà protetto da chiavi diverse (le SAK).

4.10.2.3 SAK La **SAK** (*Secure Association Key*) è la chiave *operativa* che MACsec usa sui *Secure Channel* per proteggere i frame (confidenzialità, integrità e anti-replay). Le SAK sono generate dal *Key Server* con un RNG, poi distribuite ai peer cifrandole con la KEK e firmandole con l'ICK. Ogni direzione di trasmissione usa la propria Secure Association (quindi SAK distinte per TX/RX), con contatori di pacchetto (*Packet Number*) per impedire riutilizzi (*replay*). Periodicamente, o al verificarsi di certe condizioni (es. soglie di PN), il Key Server esegue un *rekey* installando nuove SAK senza interrompere il traffico.

4.10.3 Due modalità d'uso del CAK

4.10.3.1 Static CAK La CAK è precondivisa manualmente tra i nodi (insieme al CKN). Pro: avvio semplice, ideale su link infrastrutturali stabili (switch–switch, switch–router). Contro: rotazione chiavi e governance sono manuali; non c'è un legame diretto con un'identità utente o con policy per-utente.

4.10.3.2 Dynamic CAK La CAK non si configura a mano: è derivata dal **MSK** (*Master Session Key*) rilasciato dal server RADIUS al termine dell'autenticazione 802.1X/EAP. Questo lega la CA a un'identità autenticata e consente rotazioni automatiche al rinnovo della sessione EAP. È lo schema tipico host–switch (NAC): l'host si autentica, il dispositivo di accesso e il peer ricavano localmente CAK/CKN tramite KDF, si elegge il Key Server e partono distribuzione SAK e protezione MACsec.

4.10.4 Cosa si scambiano i peer: il ciclo MKA

MKA usa messaggi **EAPOL-MKA** (*MKPDU*) periodici per mantenere la lista dei peer *vivi*, concordare chiavi e parametri, e rilevare variazioni di membership. Ogni MKPDU è protetto con un *ICV* calcolato tramite **ICK**, così che solo chi possiede la **CAK** possa partecipare.

4.10.4.1 Annuncio, stato e priorità Ad intervalli regolari, ciascun nodo invia:

- le proprie capacità *MACsec* (suite, offset di confidenzialità, supporto replay protection);
- il proprio identificatore di membro (*MI*) e contatore di messaggi (*MN*) per il controllo di liveness;
- la lista dei peer *vivi* (chi ho visto di recente) per allineare la vista di gruppo;
- la priorità di *Key Server* e parametri di tie-break (es. indirizzo MAC).

Questo scambio, protetto da ICK, impedisce a terzi di iniettare o alterare lo stato del gruppo.

4.10.4.2 Elezione del Key Server Tutti i partecipanti applicano la stessa regola di elezione (priorità amministrativa e tie-break deterministico). Il vincitore diventa **Key Server** e:

1. genera le **SAK** con un RNG robusto;
2. sceglie i parametri di uso (es. *Association Number*, finestra anti-replay);
3. distribuisce le SAK ai peer cifrandole con la KEK e firmando i messaggi con l'ICK (es. AES Key Wrap + ICV).

Il rekey può essere attivato da timer, soglie di *Packet Number*, o cambi di membership.

4.10.4.3 Apertura dei Secure Channel Installata la SAK:

- ogni direzione crea un **Secure Channel** (SC) con una o più **Secure Association** (SA), identificate da *AN*;

- il trasmettitore usa la SAK per la propria SA *TX*, i ricevitori la installano come SA *RX*;
- MACsec aggiunge il *SecTAG* e calcola l'ICV sul frame; il contatore di pacchetto (*PN*) cresce monotonicamente e abilita la protezione *anti-replay* (con finestra configurata).

4.10.4.4 Vita della sessione Gli *heartbeat* (MKPDU periodici) mantengono sincronizzati MI/MN e la lista dei peer. Se un nodo non si vede per oltre il timeout, viene rimosso dal gruppo e il Key Server invalida le SA pertinenti, avviando un *rekey* per i restanti membri. In caso di rientro o di nuovo peer, la membership viene aggiornata e si negoziano SAK fresche senza interrompere il traffico.

4.10.5 Collegamento con le policy 802.1X/NAC

802.1X/NAC decide *chi entra e con quali permessi*; **MKA/MACsec** decide *come vengono protetti i frame* sul link.

1. L'host si autentica via 802.1X/EAP; il NAC applica policy (*VLAN dinamiche, ACL/UCL* per utente/gruppo).
2. (Opzionale) In *Dynamic CAK* la **CAK** è derivata dal **MSK** della sessione EAP, legando la sicurezza L2 all'identità autenticata.
3. MKA forma la *Connectivity Association* (CKN/CAK), elegge il Key Server e distribuisce le **SAK**.
4. Da quel momento, il traffico sulla porta autorizzata è protetto da **MACsec** (confidenzialità, integrità, anti-replay) *indipendentemente* dalla VLAN o dalle ACL applicate.

In sintesi: 802.1X/NAC governa *identità e policy*; MKA/MACsec fornisce *protezione crittografica del link*. Le due componenti sono complementari e coordinate: un cambio di identità/policy (ri-autenticazione) può innescare una nuova CAK e un nuovo ciclo di SAK.

Laboratorio

4.11 Simple MKA lab with Linux

4.12 Laboratorio: MKA/MACsec su Linux (Static CAK)

4.12.1 Scopo

Configurare **MKA** (802.1X-2010) per distribuire le **SAK** di **MACsec** su un link Ethernet, creando l’interfaccia `macsec0` e verificando cifratura/integrità del traffico L2.

4.12.2 Topologia

Due host Linux collegati back-to-back (o tramite switch trasparente a L2). Ogni host esegue `wpa_supplicant` con driver `macsec_linux` per negoziare MKA.

Topology

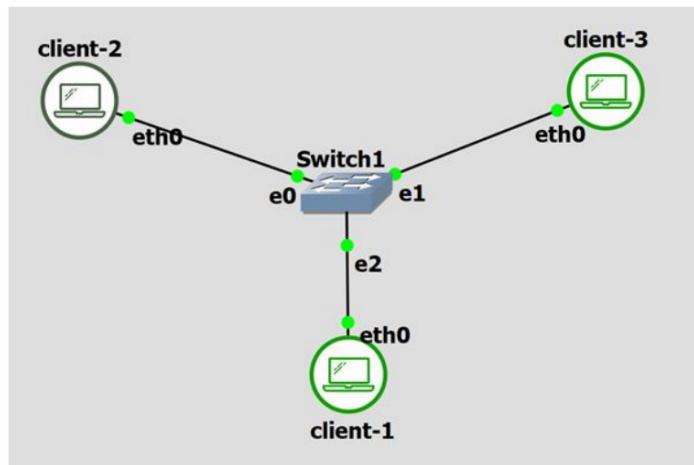


Figura 33: (topologia della rete)

4.12.3 Procedura

4.12.3.1 Generazione delle chiavi (uguali su *tutti* i peer)

```
# CAK = 16 byte esadecimali (32 hex chars)
openssl rand -hex 16
# CKN = 32 byte esadecimali (64 hex chars)
openssl rand -hex 32
```

4.12.3.2 Configurazione MKA su ciascun host

Creare `macsec.conf` (uguale su entrambi, con stessi `mka_cak` e `mka_ckn`):

```
eapol_version=3          # 802.1X-2020; richiesto per wired
ap_scan=0                 # profilo wired
network={
    key_mgmt=NONE        # wired
    eapol_flags=0         # wired
```

```

    macsec_policy=1    # abilita MACsec se possibile
    mka_cak=0123456789abcdef0123456789abcdef
    mka_ckn=00112233445566778899aabbccddeeff0011223344556677
}

```

4.12.3.3 Avvio di MKA e creazione dell'interfaccia `macsec0`

```
wpa_supplicant -i eth0 -B -D macsec_linux -c macsec.conf
```

Se l'handshake MKA va a buon fine, il kernel crea `macsec0`.

4.12.3.4 Indirizzamento IP sulla `macsec0`

```
ip addr add 10.0.1.1/24 dev macsec0    # Host A
ip addr add 10.0.1.2/24 dev macsec0    # Host B
ip link set macsec0 up
```

4.12.3.5 Test base di connettività

```
ping 10.0.1.2      # da Host A verso Host B
```

Il traffico passa attraverso `macsec0` con cifratura/integrità a L2.

4.12.4 Verifiche

4.12.4.1 Log MKA

Nei log di `wpa_supplicant` attendersi, in sequenza: *Key Server announcement* → *election process* → *SAK distribution*.

4.12.4.2 Stato interfaccia

```
ip link show macsec0
```

L'interfaccia deve essere UP e usata come device IP.

4.12.5 Troubleshooting

- **Interfaccia `macsec0` assente:** verificare `-D macsec_linux, ap_scan=0, eapol_version=3`.
- **Errore sulle chiavi:** `mka_cak` deve essere esattamente 16B (32 hex); `mka_ckn` 32B (64 hex).
- **Niente traffico IP:** assegnare l'IP su `macsec0`, non su `eth0`.
- **Mancata elezione del Key Server:** assicurarsi che entrambi i peer siano *vivi* (link up) e condividano CKN/CAK.

4.12.6 Pulizia

```
pkill wpa_supplicant
ip link del macsec0  # se presente
```

4.12.7 Estensioni & domande d'esame

4.12.7.1 Dynamic CAK (accenno)

In alternativa allo *Static CAK*, il **CAK** può essere derivato dal **MSK** ottenuto via 802.1X/EAP con backend RADIUS; il resto (elezione Key Server, distribuzione SAK) è invariato.

5 SYS_01 — Introduzione ai Security Frameworks

5.1 Quadro generale e obiettivo

La sicurezza informatica non è uno stato binario ma un processo continuo: si riduce il *costo d'attacco* mantenendo usabilità e disponibilità accettabili. Ogni strato (hardware, firmware, kernel, librerie, applicazioni) introduce superfici d'attacco e dipendenze; perciò servono principi comuni e *framework* che rendano il miglioramento *ripetibile*, *misurabile* e *auditabile*. L'obiettivo del modulo è collegare i **principi fondativi** (autenticità, autorizzazione, accountability) con i **framework operativi** (FIPS 200, CIS Controls, ISO/IEC 27000) e mostrare come si traducano in piani d'azione.

5.2 Principi fondamentali

5.2.0.1 Confidenzialità

Che cos'è:

È la proprietà per cui l'informazione è accessibile solo ai soggetti autorizzati, impedendo letture, copie o divulgazioni non consentite.

Perché serve:

Senza confidenzialità, qualsiasi misura di sicurezza perde significato: un'informazione rubata o intercettata può compromettere l'intero sistema.

Come si ottiene:

Attraverso cifratura in transito (TLS, SSH) e a riposo (Full Disk Encryption, database encryption), una gestione sicura delle chiavi (KMS, HSM), segmentazione di rete, principio del need-to-know e riduzione dei dati trattati al minimo necessario.

I Modelli sono: *simmetrica* (AES: veloce, chiave condivisa) e *asimmetrica* (RSA/EC: coppie pub/priv, abilita anche firme). Sicurezza “computazionale”: l'obiettivo è rendere impraticabile la ricerca esaustiva della chiave. La gestione delle chiavi (KMS, rotazione, *separation of duties*) è parte integrante del controllo.

Trappole comuni:

Chiavi salvate in chiaro o condivise tra utenti, backup e log non cifrati, assenza di cifratura interna su link “fidati”.

5.2.0.2 Integrità

Che cos'è:

È la garanzia che i dati o i componenti software non siano stati modificati in modo non autorizzato, oppure che ogni modifica sia tracciabile e verificabile.

Come si ottiene:

Con hash e HMAC, firme digitali, controlli di modifica (change management e principi 4-eyes), log e backup immutabili (WORM), validazioni in pipeline CI/CD.

Esempi:

Checksum end-to-end, code signing, manifest firmati, blockchain privata per log e supply-chain verification.

Attenzioni:

CRC o hash non firmati non offrono sicurezza reale; le chiavi di firma vanno custodite e ruotate; occorre distinguere tra collisione e preimage per valutare la robustezza di un algoritmo.

5.2.0.3 Disponibilità

Che cos'è:

È la capacità di mantenere servizi e dati accessibili con prestazioni accettabili, anche in presenza

di guasti o attacchi.

Come si ottiene:

Con ridondanza (N+1, active-active), strategie di degrado controllato (graceful degradation), meccanismi di protezione come circuit breaker, capacity planning, autoscaling e piani di continuità (DR/BCP).

Minacce tipiche:

Attacchi DoS/DDoS, saturazione delle risorse, errori di configurazione, catene di dipendenze non ridondante, aggiornamenti mal pianificati.

5.2.0.4 Autenticità e Autorizzazione

Autenticità — che cos’è:

È la garanzia che l’identità dichiarata da un soggetto corrisponda effettivamente a chi afferma di essere.

Autenticità — come si ottiene:

Tramite meccanismi di autenticazione a più fattori (MFA), gestione sicura delle credenziali, certificati digitali, attestazione hardware e protocolli di handshake sicuro.

Autorizzazione — che cos’è:

È il processo di assegnare o negare risorse a un soggetto autenticato in base al suo ruolo, contesto o policy.

Autorizzazione — come si ottiene:

Applicando il principio del least privilege, modelli RBAC (Role-Based Access Control) o ABAC (Attribute-Based), segregazione dei doveri, accessi JIT/JEA e revisioni periodiche dei permessi.

Pattern utili:

SSO centralizzato, token scadibili, sessioni brevi, meccanismi di break-glass tracciato per accessi di emergenza.

5.2.0.5 Accountability

Che cos’è:

È la possibilità di attribuire in modo affidabile ogni azione compiuta in un sistema a un soggetto identificabile.

Come si ottiene:

Attraverso logging centralizzato e immutabile (SIEM), sincronizzazione oraria precisa (NTP/PTP), integrità dei log (HMAC, firme o catene di hash), correlation ID end-to-end e una catena di custodia documentata per la forensica.

Attenzioni:

Bilanciare privacy e obblighi di tracciabilità; proteggere sia il canale di logging sia il repository; stabilire tempi di retention coerenti con GDPR e compliance interna.

5.2.0.6 Dipendibilità

Che cos’è:

È la qualità complessiva di un sistema nel fornire un servizio affidabile, sicuro, mantenibile e resiliente nel tempo.

Come si ottiene:

Attraverso osservabilità (metriche, log, tracing), deploy sicuri (feature flags, canary release, rollback), chaos engineering, e post-incident review per il miglioramento continuo.

Connessioni:

La CIA tutela dati e servizi, autenticità e autorizzazione controllano chi accede, accountability ne permette la verifica, e la dipendibilità orchestra tutto il ciclo di vita — dalla progettazione, all’esercizio, fino alla risposta agli incidenti.

5.3 Perché servono i Security Frameworks

La sicurezza non può più basarsi solo su “buone pratiche” isolate: i sistemi moderni sono troppo complessi, distribuiti e interdipendenti. Senza una struttura comune, ogni reparto o fornitore valuterebbe i rischi in modo diverso, rendendo impossibile il confronto e la gestione coerente della sicurezza.

I **Security Frameworks** servono proprio a questo: forniscono un linguaggio condiviso, un insieme ordinato di controlli riusabili e una base di confronto per misurare maturità e copertura. Ogni framework traduce i principi astratti (come confidenzialità, integrità e disponibilità) in **controlli pratici**, assegnando priorità e responsabilità operative. Inoltre, introducono un metodo sistematico di miglioramento continuo basato sul ciclo *Plan–Do–Check–Act* (PDCA): pianificare i controlli, attuarli, verificarne l’efficacia e correggere le lacune.

In sintesi, i framework rendono la sicurezza:

- **misurabile**, grazie a controlli e metriche standard;
- **ripetibile**, perché le procedure possono essere replicate e auditabili;
- **comunicabile**, poiché tecnologia, processi e persone usano la stessa terminologia;
- **migliorabile**, grazie al monitoraggio continuo del rischio residuo.

5.4 Tre famiglie a confronto

5.4.1 FIPS 200 — Requisiti minimi per i sistemi federali

Il **FIPS 200** (Federal Information Processing Standard) definisce i requisiti minimi di sicurezza per i sistemi informativi delle agenzie federali statunitensi. È un approccio prescrittivo e normativo: stabilisce quali controlli devono essere presenti e come verificarne l’attuazione.

Le principali famiglie di controllo comprendono: *Access Control, Audit and Accountability, Configuration Management, Contingency Planning, Identification and Authentication, Incident Response, Maintenance, Risk Assessment* e molte altre.

Ogni organizzazione deve:

- valutare periodicamente i controlli implementati;
- pianificare e attuare azioni correttive;
- ottenere l’autorizzazione all’operatività dei propri sistemi (*Authorization to Operate, ATO*);
- mantenere un monitoraggio continuo dell’efficacia delle misure.

Il FIPS 200 è quindi una **baseline di conformità** utile dove esistono vincoli regolatori o contrattuali (es. ambienti governativi o fornitori di enti pubblici). Fornisce un modello forte in termini di *assurance* e tracciabilità dei controlli, ma poco flessibile in contesti dinamici o non federali.

5.4.2 CIS Critical Security Controls — L’igiene prioritaria

I **CIS Controls** (Center for Internet Security) rappresentano un insieme pratico e priorizzato di buone pratiche operative, pensato per essere attuabile rapidamente anche da organizzazioni non grandi.

I controlli sono 20 (raggruppati in famiglie come *Inventory and Control, Secure Configuration, Continuous Vulnerability Management, Controlled Use of Administrative Privileges, Malware Defense, Incident Response...*), ciascuno articolato in sotto-controlli concreti.

I primi cinque sono considerati la “**cyber hygiene di base**”, cioè le difese fondamentali che riducono fino all’85% delle minacce comuni:

1. inventario di dispositivi autorizzati e non autorizzati;
2. inventario del software installato;
3. configurazioni sicure di host e apparati;
4. gestione continua delle vulnerabilità e delle patch;
5. uso controllato dei privilegi amministrativi.

La forza dei CIS Controls sta nella loro concretezza: sono pensati per rispondere alla domanda “*da dove cominciare domani mattina?*”. Sono verificabili, misurabili e fortemente orientati all’operatività quotidiana.

Per questo motivo vengono spesso adottati come base operativa su cui innestare framework più ampi (ISO, NIST, ecc.), fungendo da **checklist pragmatica** per il miglioramento rapido della postura di sicurezza.

5.4.3 ISO/IEC 27000 — ISMS e gestione del rischio

La famiglia **ISO/IEC 27000** fornisce un modello internazionale per la gestione della sicurezza delle informazioni basato sull’approccio **ISMS** (*Information Security Management System*).

Il cuore della famiglia è la **ISO/IEC 27001**, che definisce i requisiti per stabilire, mantenere e migliorare un ISMS certificabile. La **27002** fornisce linee guida e buone pratiche per l’attuazione dei controlli, mentre la **27005** approfondisce la gestione del rischio.

Esistono inoltre estensioni verticali per contesti specifici:

- **27017** e **27018** per la sicurezza e privacy nel cloud;
- **27019** per i sistemi industriali (ICS/SCADA);
- **27011** (ITU-T X.1051) per operatori telco.

Il modello ISO si basa su un approccio **risk-based** e adattabile: ogni organizzazione identifica e valuta i propri rischi, seleziona i controlli più adatti (dall’*Annex A*) e li integra nei processi aziendali secondo il ciclo PDCA.

L’obiettivo non è la “sicurezza assoluta”, ma la **gestione consapevole del rischio**, bilanciando protezione, costi e conformità. Il valore aggiunto della serie ISO 27000 è la sua scalabilità e certificabilità: consente audit esterni, comparabilità e riconoscimento internazionale, fungendo da ponte tra *compliance* e *governance*.

Sintesi comparativa

- **FIPS 200**: prescrittivo, orientato alla conformità e all’accreditamento dei sistemi federali. Ottimo per la *baseline* normativa, poco flessibile fuori da contesti regolati.
- **CIS Controls**: operativo e immediato, fornisce priorità chiare e controlli attuabili. Ideale per migliorare rapidamente l’igiene di sicurezza.
- **ISO/IEC 27000**: gestionale e certificabile, integra sicurezza, rischio e governance. Indicato per aziende che mirano a maturità e riconoscimento formale.

Molte organizzazioni combinano i tre approcci: i *CIS Controls* come base operativa, la *ISO 27001* come cornice di governance, e il *FIPS 200* come riferimento di conformità in ambiti regolati.

6 SYS_02 — Hardware Primer

6.1 Moore's Law e l'evoluzione del calcolo

Nel 1965 Gordon Moore osservò che il numero di transistor per unità di area nei circuiti integrati tendeva a raddoppiare ogni due anni. Questa legge empirica — *Moore's Law* — ha guidato per decenni l'intera industria dei semiconduttori: più transistor significano clock più alti, parallelismo maggiore e costi inferiori per unità di potenza computazionale. Per quasi quarant'anni il trend è stato rispettato, permettendo un'esplosione di potenza e miniaturizzazione, ma a partire dai primi anni 2000 sono emersi vincoli fisici e termici noti come **power wall**: oltre i ~ 130 W di dissipazione termica per CPU, i guadagni marginali diventano insostenibili.

6.2 Fine della crescita lineare e parallelismo

La stagnazione delle frequenze e la saturazione del numero di transistor utili (come mostrato da Hennessy e Patterson nella Turing Lecture 2018) hanno spinto verso architetture parallele: più core, pipeline più profonde e ottimizzazioni speculative. Amdahl, già nel 1967, formalizzò il limite teorico di tale approccio: il guadagno complessivo $S_{latency}$ da un'ottimizzazione parziale dipende dalla frazione p di codice migliorato e dal relativo speedup s , secondo

$$S_{latency} = \frac{1}{(1-p) + \frac{p}{s}}$$

da cui risulta chiaro che, se solo una piccola parte del programma è parallelizzabile, il beneficio totale resta limitato.

6.3 Pipeline e architettura superscalare

Per superare i limiti dei primi core multicycle (lenti ma complessi), negli anni '80 si introdusse il **pipelining**: un'istruzione viene divisa in più fasi temporali (fetch, decode, execute, write-back), permettendo l'esecuzione "a catena". Idealmente il CPI (Cycles Per Instruction) diventa 1, ma solo se non vi sono conflitti o branch. Negli anni '90 arrivarono le **architetture superscalari**, capaci di eseguire più istruzioni contemporaneamente grazie a unità funzionali replicate e scheduling dinamico a runtime: il processore decide in tempo reale quali istruzioni emettere in parallelo, massimizzando il throughput.

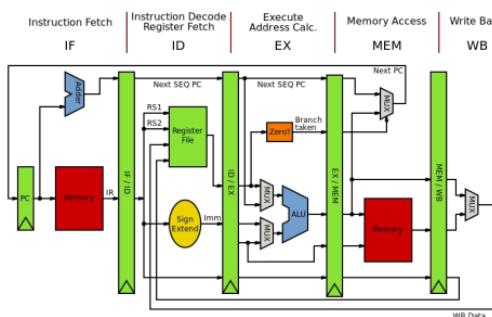


Figura 34: Architettura pipeline MIPS

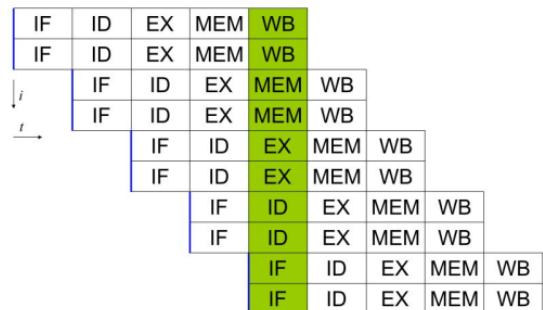


Figura 35: Esecuzione parallela di più istruzioni sulla stessa CPU

6.4 Branch Prediction: principi e strategie

La predizione dei salti è una delle ottimizzazioni più importanti per mantenere il *throughput* della pipeline: ogni salto condizionale interrompe il flusso sequenziale di istruzioni e può costringere a svuotare (flush) la pipeline se la direzione effettiva non è quella prevista. Con pipeline sempre

più profonde e architetture superscalari che eseguono molte istruzioni per ciclo, il costo di una **misprediction** può arrivare a decine di cicli.

6.4.0.1 Perché serve una predizione Quando la CPU incontra un’istruzione di salto (**branch**), non può sapere immediatamente se il salto sarà preso (*taken*) o meno (*not taken*) finché il confronto logico non viene risolto nello stadio di esecuzione. Attendere significherebbe fermare la pipeline: per questo, la CPU **indovina** il risultato e continua a caricare istruzioni dal percorso ipotizzato.

6.4.0.2 Tipi di predizione

- **Statica:** la decisione è determinata a compile-time. È semplice e non richiede hardware dedicato: ad esempio, si assume che i salti in avanti (come in un **if**) non siano presi, mentre quelli all’indietro (come nei loop) lo siano. Alcuni compilatori o kernel possono anche inserire “hint” (es. attributi **likely/unlikely** o prefissi nel microcodice) per guidare il processore.
- **Dinamica:** la CPU apprende dal comportamento passato del programma, usando **storia dei branch** e piccole strutture hardware che registrano se l’ultimo salto da un certo indirizzo è stato preso o meno. La predizione viene aggiornata a runtime, quindi si adatta a comportamenti ricorrenti del software.

6.4.0.3 Contatore a 2 bit saturante Il modello più classico di predittore dinamico è il **2-bit saturating counter**: ogni branch ha un contatore a due stati fortemente orientati (00 = “fortemente non preso”, 11 = “fortemente preso”). Il contatore si incrementa o decrementa in base all’esito reale, ma non passa da un estremo all’altro in un solo errore, riducendo così le oscillazioni casuali. Questo approccio funziona bene per branch regolari, ma fallisce con strutture di controllo più complesse come **loop annidati**, dove il contesto del branch esterno viene continuamente “sporcato” da quello interno.

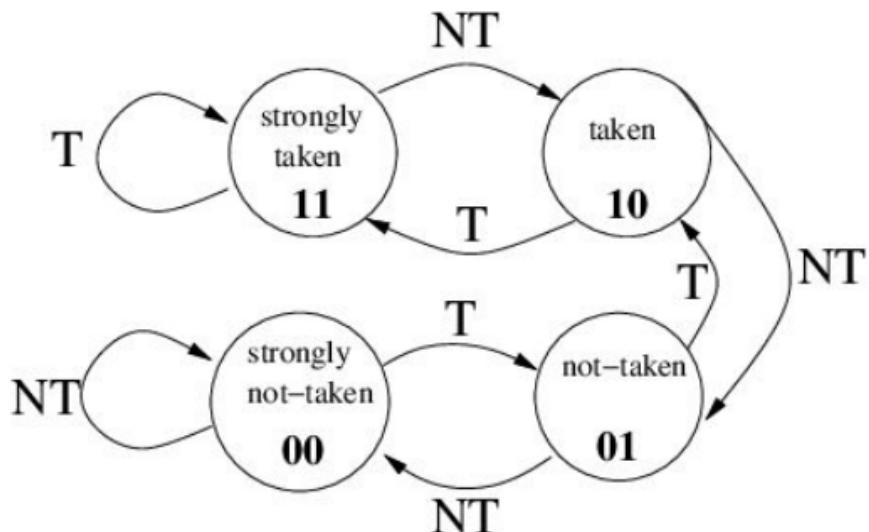


Figura 36: 2 bit saturating counter

6.4.0.4 Predittori correlati e multilivello Per migliorare l’accuratezza, i processori moderni impiegano predittori che tengono conto non solo della storia di un singolo branch, ma anche delle correlazioni tra branch consecutivi:

- **Correlated (two-level) predictor:** mantiene una **Global History Register (GHR)** che codifica l'esito degli ultimi m branch (presa = 1, non presa = 0). Il valore di questo registro viene usato come indice in una **Pattern History Table (PHT)**, che contiene i contatori a 2 bit. In questo modo la CPU “riconosce” sequenze ricorrenti di salti.

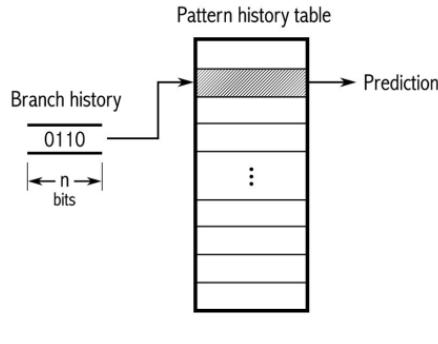


Figura 37: Pattern History Table

- **Hybrid o Tournament predictor:** combina più predittori (ad esempio uno locale e uno globale) e sceglie dinamicamente quale usare in base alla precisione recente. Una piccola tabella di selezione con contatori a 2 bit tiene traccia di quale predittore ha vinto più spesso per un certo branch.

Un esempio celebre è quello del processore **DEC Alpha 21264**, che impiegava un algoritmo di predizione “tournament” con meno dell’1% di mispredizioni totali, occupando appena il 2% dell’area del chip.

6.4.0.5 Ottimizzazioni hardware Oltre ai predittori veri e propri, esistono altre strutture dedicate a ridurre il tempo di risoluzione del salto:

- **Branch Target Buffer (BTB):** una piccola cache che associa a ciascun indirizzo di branch il suo target previsto. Se il salto è predetto “taken”, il BTB fornisce immediatamente il nuovo indirizzo di fetch, senza attendere l’ALU.
- **Return Address Stack (RAS):** uno stack hardware che salva l’indirizzo di ritorno delle chiamate a funzione (`call/return`). Poiché la maggior parte dei salti indiretti sono proprio ritorni da funzione, questo migliora drasticamente la predizione nei linguaggi ad oggetti e nei programmi modulari.
- **Fetch both targets:** alcune architetture ad alte prestazioni (es. mainframe IBM) prelevano simultaneamente le istruzioni da entrambi i possibili percorsi (taken e not-taken), per poi scartare quello errato. È una soluzione costosa in termini di banda e cache, ma riduce quasi a zero la penalità di mispredizione.

6.4.0.6 Importanza crescente della predizione Con pipeline sempre più profonde e front-end superscalari, ogni ciclo perso per un salto errato può comportare la perdita di decine di istruzioni. La branch prediction moderna è quindi un componente centrale della microarchitettura: la precisione di questi predittori incide direttamente sul consumo energetico, sull’IPC (*Instructions Per Cycle*) e sulla sicurezza — come dimostrato dalle vulnerabilità speculative (*Spectre, Meltdown*) che sfruttano proprio l’esecuzione speculativa e il caching dei risultati predetti.

6.5 Simultaneous Multithreading (SMT)

Con la stagnazione del clock e i limiti di dissipazione, si è puntato a sfruttare meglio le risorse interne del core. Lo **SMT** consente a un singolo core fisico di apparire come più core logici: ogni thread mantiene il proprio stato architettonico (registri, flag, PC), ma condivide le unità funzionali. In caso di stallo di un thread (es. cache miss), un altro può utilizzare immediatamente le pipeline inutilizzate. Intel implementa SMT come **Hyper-Threading**: due thread logici per core, con alternanza in caso di contesa.

6.6 Pipeline interna nei processori Intel Xeon

La pipeline dei Xeon è divisa in due macro-stadi:

- **Front End:** decodifica le istruzioni CISC x86 in micro-operazioni RISC (μops) tramite un microcode ROM e le memorizza nella **Trace Cache (TC)**. La TC conserva le μops nel loro ordine di esecuzione effettiva, evitando di ridurre in continuazione istruzioni già viste.
- **Out-of-Order Engine:** gestisce l'esecuzione delle micro-operazioni (μops) fuori ordine rispetto al flusso originale, massimizzando il parallelismo interno. Dopo il **μop Queue**, le istruzioni entrano nello stadio di **Register Rename**, dove i registri logici vengono mappati su registri fisici per eliminare conflitti (false dipendenze). Le μops vengono poi inserite in una **coda di scheduling** che decide, a runtime, quali operazioni emettere in base alla disponibilità delle risorse e dei dati. Una volta pronti gli operandi, il blocco **Execute** effettua le operazioni reali, accedendo alla **L1 Data Cache** per letture e scritture. I risultati vengono scritti nei registri fisici e infine ritirati in ordine logico dal **Reorder Buffer (ROB)**, che ricostruisce la sequenza originale del programma garantendo coerenza architettonica. La figura 38 mostra la sequenza interna del motore Out-of-Order nei processori Xeon.

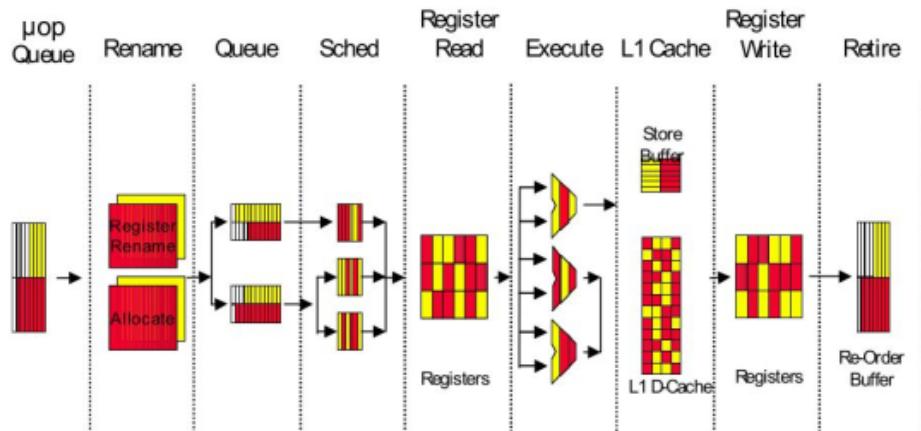


Figura 38: Pipeline interna Out-of-Order nei processori Xeon: le μops attraversano gli stadi di rinomina, scheduling, esecuzione e *retirement* nel **Reorder Buffer**, che garantisce il commit in ordine architettonico.

Un *Trace Cache hit* consente al core di servire μops senza passare per la decodifica, mentre un miss richiede fetch e decode da L2 (ovvero il livello 2 della cache). Questo modello riduce il collo

di bottiglia di decodifica tipico dell'ISA x86, ottenendo parallelismo interno senza moltiplicare il numero di core.

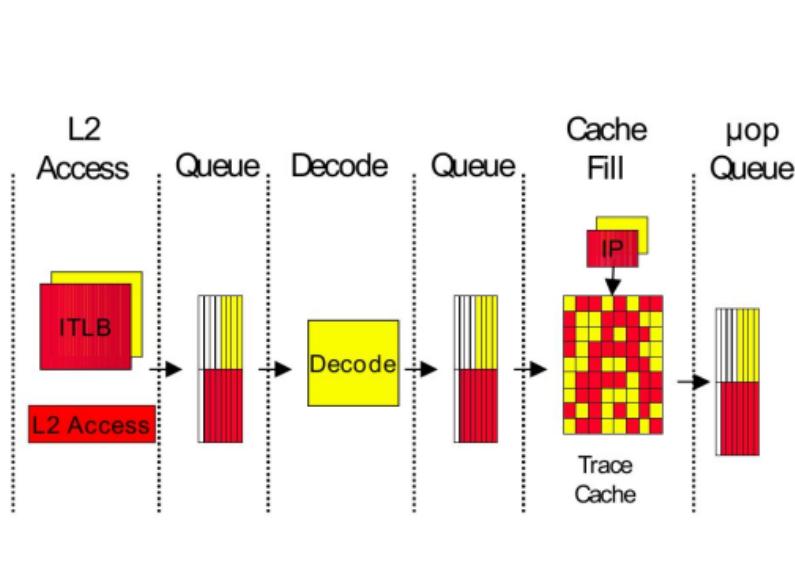


Figura 39: Flusso del **front-end** nei processori Intel: le istruzioni x86 vengono prelevate dalla cache L2, decodificate in micro-operazioni (*μ ops*) e memorizzate nella **Trace Cache**. In caso di *hit*, le *μ ops* vengono fornite direttamente alla pipeline senza ripetere il ciclo di fetch e decode.

6.7 Gerarchia di memoria

La distanza prestazionale tra CPU e memoria principale (DRAM) cresce di circa il 50% l'anno: per colmare il gap si adotta una **memoria gerarchica** (L1, L2, L3, RAM, disco). Ogni livello inferiore è più capiente ma più lento e meno costoso per bit. Il processore interagisce direttamente solo con L1: un **cache hit** consente accesso immediato, mentre un **miss** causa richieste ricorsive ai livelli successivi fino alla RAM (da L1 vado a L2, se c'è hit copio dato in L1, altrimenti vado in L3 e vedo hit/miss e così via).

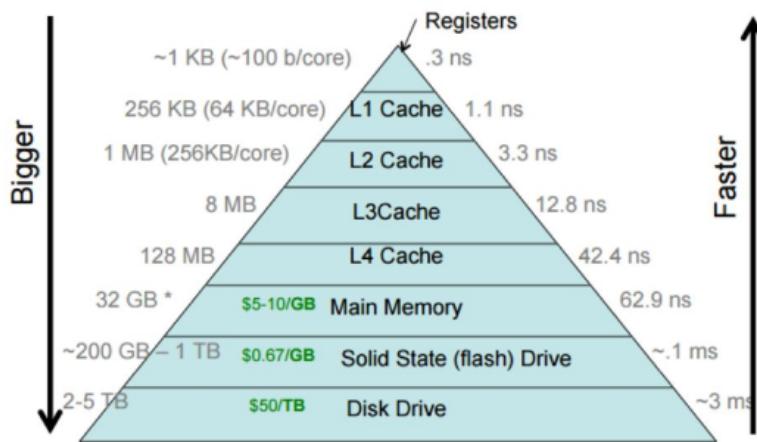


Figura 40: Schema piramidale della gerarchia di memoria

6.7.0.1 Inclusività e mappatura Inclusività tra livelli di cache. Nelle gerarchie moderne (L1, L2, L3) ogni livello può decidere se mantenere o meno una copia dei dati già

presenti nei livelli inferiori. Si distinguono tre politiche principali:

- **Inclusive:** tutti i blocchi presenti in L1 devono esistere anche in L2 (e, se presente, in L3). Ciò semplifica la **coerenza**: se un core invalida un dato in L3, si sa che anche le sue copie nelle cache inferiori devono essere invalidate. Tuttavia, l'inclusività riduce la capacità effettiva complessiva (L2 “spreca” spazio per dati già duplicati).
- **Exclusive:** i blocchi si trovano in un solo livello alla volta — L1 e L2 contengono set disgiunti. Aumenta la capacità totale utile, ma complica la gestione: un miss in L1 può richiedere di “spostare” un blocco da L2 a L1, generando più traffico interno.
- **Non-inclusive / Non-exclusive (NINE):** via di mezzo più flessibile; non impone regole rigide di duplicazione ma lascia la scelta all’hardware. È usata nei processori recenti (es. AMD Ryzen) per bilanciare prestazioni e flessibilità.

Politiche di mappatura. Quando la CPU deve memorizzare un blocco di memoria nella cache, serve una regola per stabilire in quale linea (entry) posizionarlo. Questa scelta influenza direttamente le prestazioni e il tasso di *miss*.

- **Direct-mapped cache.**

Ogni indirizzo di memoria corrisponde a una sola posizione possibile in cache, determinata da alcune bit dell’indirizzo (funzione modulo del numero di linee). È la struttura più semplice e veloce: un solo comparatore e accesso deterministico. Tuttavia, se due blocchi diversi mappano sulla stessa linea, si generano **conflitti ripetuti** (cache thrashing). È adatta a cache piccole e a livello L1, dove si privilegia la latenza minima.

- *Pro:* accesso veloce, logica semplice, basso consumo.
- *Contro:* tasso di miss elevato in presenza di conflitti.

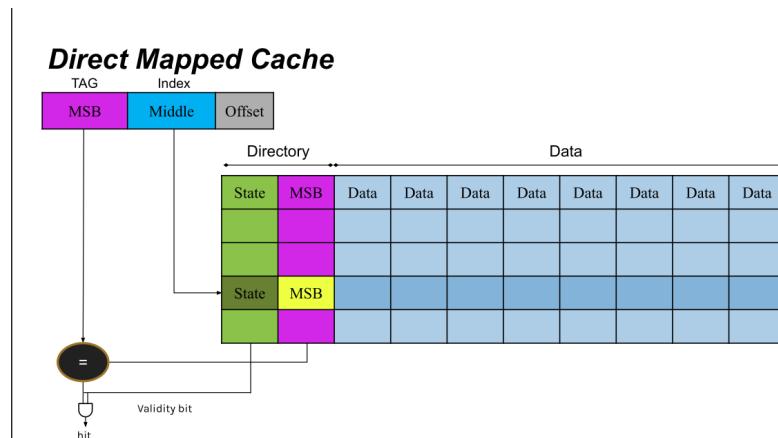


Figura 41: Direct mapped cache

- **Fully associative cache.**

Un blocco di memoria può essere collocato in *qualsiasi* linea della cache. L’indirizzo viene confrontato con tutte le etichette (tag) in parallelo per verificare un hit. Offre la massima flessibilità e il minor numero di conflitti, ma richiede **comparazioni parallele costose** in termini di area e potenza. È usata tipicamente per cache molto piccole o buffer speciali (es. TLB).

- *Pro:* minimi miss da conflitto, utilizzo ottimale dello spazio.
- *Contro:* costosa e lenta da scalare, elevato consumo energetico.

Fully Associative Cache: read access

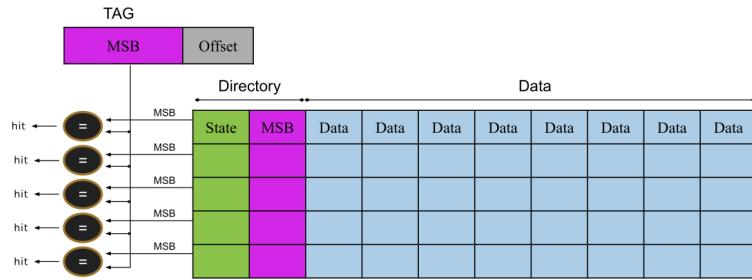


Figura 42: Fully associative cache

- **Set-associative cache.**

È il compromesso più comune. La cache è divisa in gruppi (set) di n linee ciascuno; un indirizzo mappa su un set specifico (tramite parte dell'indirizzo), ma può occupare *una qualsiasi* delle n linee di quel set. Il grado di associazione (2-way, 4-way, 8-way, ...) determina quanti blocchi possono coesistere nello stesso set. L'hardware esegue fino a n confronti paralleli, mantenendo un buon bilancio tra velocità e flessibilità.

- *Pro:* riduce fortemente i miss da conflitto rispetto al direct-mapped, con costo contenuto.
- *Contro:* leggermente più lenta e più complessa; aumento di consumo con l'associatività.

4-way Set Associative Cache: read access

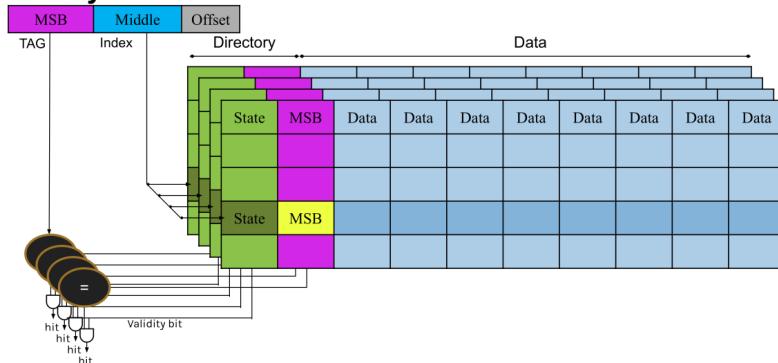


Figura 43: 4way set Associative cache

Sintesi pratica.

- L1 è spesso **direct-mapped o 2-way** per minimizzare la latenza.
- L2 e L3 sono di solito **8-way o più** per massimizzare la capacità.
- I processori moderni (Intel/AMD) adottano strutture **inclusive o NINE** per semplificare la coerenza cache tra core.

6.7.0.2 Sostituzione ed aggiornamento Le politiche di rimpiazzamento (cambio dei blocchi di cache) determinano quale blocco deve essere espulso quando lo spazio di memoria nella cache è pieno. Le politiche più comuni sono:

- **LRU (Least Recently Used):** il blocco meno recentemente utilizzato viene espulso. Questo approccio è basato sull'assunzione che i blocchi recentemente usati abbiano più probabilità di essere riutilizzati rispetto a quelli che non sono stati usati di recente. È uno dei metodi più efficaci, ma richiede la gestione di un registro dei tempi di accesso per ogni blocco, il che può aumentare la complessità hardware.
- **FIFO (First In, First Out):** il blocco che è stato caricato per primo nella cache è quello che viene espulso. È semplice da implementare, ma meno ottimizzato rispetto a LRU, poiché non tiene conto dell'effettivo utilizzo recente dei dati.
- **Random:** espelle un blocco scelto a caso. Questo metodo è semplice da implementare e veloce, ma può comportare una maggiore probabilità di espulsione di blocchi che saranno riutilizzati di frequente, riducendo l'efficienza della cache.
- **Round Robin:** utilizza un ciclo continuo per scegliere quale blocco espellere, utilizzando un puntatore che viene aggiornato a ogni rimpiazzo. Questo approccio è meno comune ma può essere utilizzato in alcune cache completamente associative.

Per quanto riguarda le scritture nelle cache, esistono due principali strategie di gestione:

- **Write-through:** ogni volta che viene eseguita una scrittura nella cache, la stessa scrittura viene immediatamente propagata nella memoria principale (o nella cache di livello superiore). Questo approccio è semplice da implementare, ma comporta un rallentamento delle prestazioni, in quanto ogni scrittura nella cache necessita anche di una scrittura nella memoria principale, aumentando il traffico sulla memoria.
- **Write-back:** le scritture vengono effettuate solo nella cache, e i dati vengono trasferiti alla memoria principale solo quando il blocco viene rimosso dalla cache, cioè al momento della sostituzione. Questo approccio riduce il traffico di scrittura nella memoria principale e quindi migliora le prestazioni, ma è più complesso da gestire, in quanto è necessario tenere traccia dei blocchi modificati nella cache.

La scelta tra queste due strategie dipende dalle specifiche esigenze di prestazione e dalla complessità del sistema: **write-through** è più semplice e garantisce la coerenza immediata tra cache e memoria, mentre **write-back** è più efficiente ma richiede un meccanismo di aggiornamento ritardato, aumentando la complessità nella gestione della coerenza dei dati.

6.8 Cache coherence nei multicore

Nei sistemi multicore, ogni core dispone di una cache privata: per evitare inconsistenze sui dati condivisi serve un **protocollo di coerenza**. La coerenza (CC, Cache Coherence) garantisce che tutti i core osservino la stessa sequenza logica di aggiornamenti per una cella di memoria condivisa. Il caso ideale — **strong coherence** — impone che tutte le letture e scritture siano viste nello stesso ordine da tutti i core. Al contrario, con **weaker coherence**, le operazioni di scrittura possono non essere immediatamente visibili dagli altri core. Questo approccio riduce il carico di gestione della coerenza, permettendo al processore di eseguire altre operazioni in parallelo senza dover aspettare che tutte le scritture siano propagate e visibili a tutti i core.

Condizioni sufficienti:

1. **Single-Writer/Multiple-Readers (SWMR):** in un dato istante un solo core può scrivere su una cella, ma più core possono leggerla.
2. **Data-Value (DV):** il valore della cella è identico per tutti i core all'inizio di ogni epoca.

Molti sistemi reali adottano varianti **weaker coherence**, sacrificando temporaneamente la visibilità per migliorare prestazioni e parallelismo.

6.8.0.1 Protocolli di coerenza In un sistema multi-core, i dati condivisi devono essere coerenti tra le varie cache locali (L1, L2) per evitare letture di valori obsoleti o incoerenti. Esistono due principali categorie di protocolli per garantire la coerenza:

- **Invalidation:** quando un core scrive in un blocco di memoria, invalida tutte le copie di quel blocco nelle altre cache. Questo significa che ogni altro core che ha una copia del blocco dovrà ricaricarlo dalla memoria principale o dal livello superiore della cache prima di poterlo usare di nuovo. L'Invalidation è un approccio semplice, che riduce il traffico di coerenza, ma può aumentare la latenza se i dati invalidati sono frequentemente richiesti.
 - *Pro:* riduce il traffico di coerenza, adatto per carichi di lavoro con bassa contesa.
 - *Contro:* può causare elevata latenza in caso di **cache miss**, poiché le copie devono essere ricaricate dalla memoria.
- **Update:** quando un core scrive un dato, aggiorna tutte le copie di quel blocco nelle altre cache. Questo approccio garantisce che ogni copia in cache sia aggiornata in tempo reale, riducendo il rischio di letture obsolete, ma comporta un maggiore traffico di coerenza tra i core.
 - *Pro:* garantisce che tutti i core vedano subito i dati più recenti.
 - *Contro:* aumenta il traffico di coerenza e può ridurre le prestazioni, specialmente in sistemi con molteplici core che scrivono frequentemente.

Implementazioni dei protocolli: I protocolli di coerenza possono essere implementati in due principali modalità:

- **Snooping:** ogni controller di cache “sente” (snoops) le richieste di accesso sulla linea di comunicazione condivisa tra i core, e reagisce di conseguenza. Ogni core monitora il bus di comunicazione (ad esempio il **system bus**) per rilevare accessi a blocchi condivisi e determinare se deve aggiornare o invalidare una cache. Questo approccio è tipico nelle architetture con un bus condiviso, come quelle usate nei sistemi a singolo socket o con numero limitato di core.
 - *Pro:* semplice da implementare, non richiede una struttura centralizzata.
 - *Contro:* inefficiente per sistemi con molti core, poiché ogni cache deve monitorare l'intero bus, aumentando il carico e il traffico.

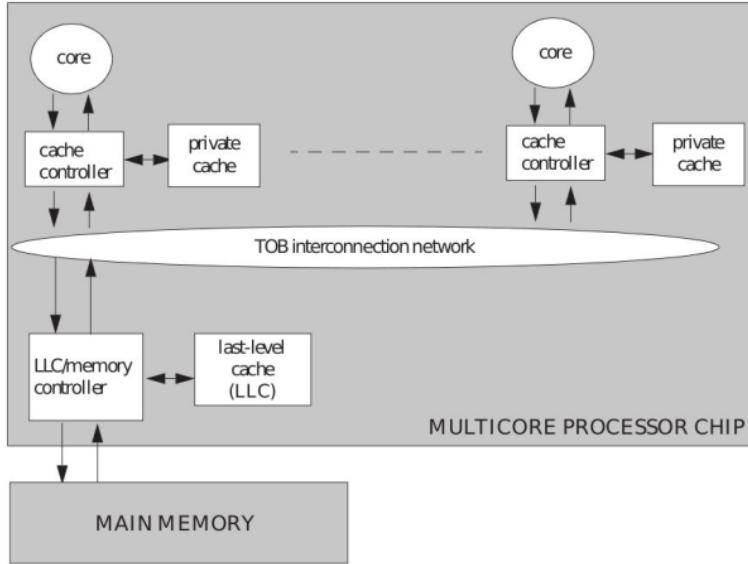


Figura 44: Snooping cache system model

- **Directory-based:** in questo modello esiste un nodo centrale, una **directory**, che tiene traccia di quali core possiedono una copia di ogni blocco di memoria. La directory riceve richieste di accesso da parte dei core e, quando un core scrive su un blocco, essa aggiorna o invalida le copie nelle altre cache. Questo modello è più scalabile rispetto allo snooping, poiché centralizza la gestione della coerenza, evitando che ogni cache debba monitorare il bus.
 - *Pro:* più scalabile in sistemi con molti core, riduce il traffico di coerenza.
 - *Contro:* maggiore complessità nell'implementazione e potenziale punto di congestione nella directory.

Sintesi:

- Il modello **Invalidate** è efficace in scenari con bassa contesa ma può rallentare l'accesso in caso di cache miss.
- Il modello **Update** è più costoso in termini di traffico di coerenza ma offre una visione più coerente dei dati.
- Il modello **Snooping** è semplice da implementare ma non scala bene per sistemi a molti core, mentre **Directory-based** è scalabile ma più complesso da gestire.

6.9 Protocolli MOESI e VI

I protocolli di coerenza sono descritti come **macchine a stati finiti** (FSM), dove ciascun blocco in cache può trovarsi in diversi stati di validità. Il protocollo più semplice è il **VI**, con due soli stati (Valid/Invalid).

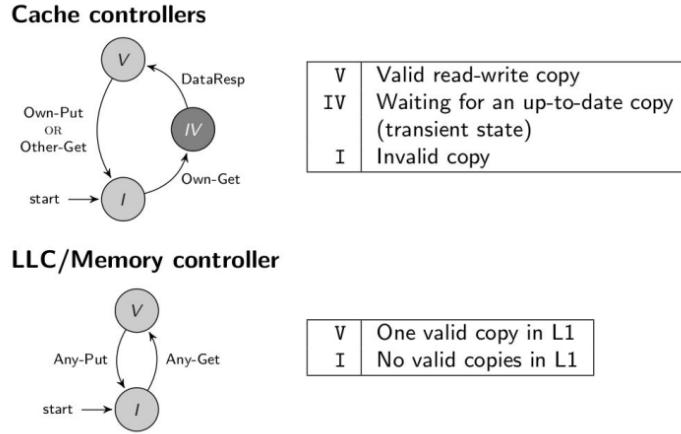


Figura 45: The vi protocol

Tuttavia, i sistemi moderni adottano versioni più espressive come il **MOESI**, che introduce:

- **M (Modified):** il blocco è valido, esclusivo, potenzialmente sporco; solo questa cache lo possiede.
- **O (Owned):** valido e potenzialmente sporco ma condiviso; la cache risponde alle richieste di altri core.
- **E (Exclusive):** valido e pulito; nessun'altra cache lo possiede.
- **S (Shared):** valido e condiviso, non modificabile.
- **I (Invalid):** non contiene dati validi.

Il protocollo MOESI consente di minimizzare traffico e latenza mantenendo comunque una visione coerente dei dati, sfruttando il concetto di *ownership* per coordinare chi deve rispondere a richieste remote.

6.9.0.1 Virtual vs. Physical Cache Indexing Le cache possono essere **indicizzate** usando l'indirizzo virtuale o quello fisico, a seconda del livello e della microarchitettura. La scelta influisce su latenza, complessità hardware e rischi di inconsistenza tra TLB e cache.

- **Virtual-indexed caches:** usano l'indirizzo virtuale generato dalla CPU, prima della traduzione tramite il **Translation Lookaside Buffer (TLB)**. Questo approccio è tipico della **L1 cache**, dove la priorità è la *bassa latenza* e la velocità di accesso supera i rischi di incoerenza. Tuttavia, poiché la traduzione virtuale→fisica non è ancora disponibile, il sistema deve gestire eventuali aliasing (lo stesso indirizzo fisico visto come virtuale diverso da più processi).
- **Physically-indexed caches:** usano invece l'indirizzo fisico, cioè quello tradotto dal TLB. Questo evita i problemi di aliasing e garantisce coerenza tra processi e livelli di cache diversi, ma richiede che la traduzione sia già avvenuta, aumentando leggermente la latenza. Le **L2 e L3 cache** usano quasi sempre indicizzazione fisica.
- **TLB (Translation Lookaside Buffer):** memorizza le traduzioni virtuale→fisico più recenti, riducendo il costo di accesso alla memoria principale. Quando il TLB fornisce rapidamente la traduzione, l'indicizzazione fisica non penalizza eccessivamente le prestazioni.

In sintesi, la cache L1 privilegia l'accesso immediato e può usare indirizzi virtuali, mentre i livelli esterni (L2, L3) scelgono la coerenza globale e usano indirizzi fisici. Questo bilanciamento permette di minimizzare la latenza mantenendo consistenza tra processi, TLB e memoria principale.

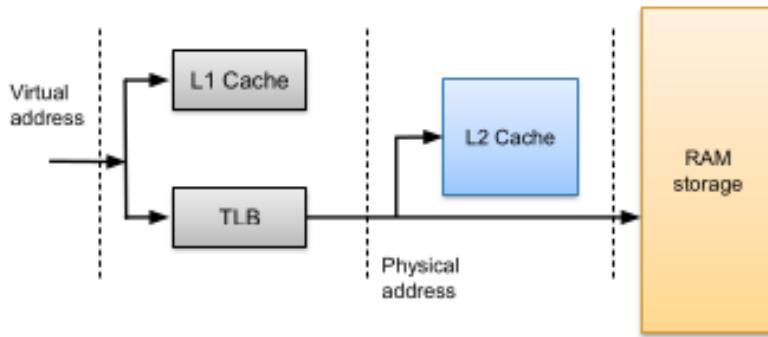


Figura 46: Virtual Vs Physical cache indexing

6.10 Hardware Transactional Memory (HTM)

Per ridurre la complessità della sincronizzazione esplicita tra thread (lock, mutex), le CPU moderne supportano la **Transactional Memory** hardware, introdotta da Intel nel 2013 (TSX). Un blocco di codice viene eseguito come transazione: tutte le operazioni sono provvisorie fino al commit, e se si verifica un conflitto con altri thread, la transazione viene annullata automaticamente. Questo approccio fornisce concorrenza “lock-free” e semplifica il codice multithreaded.

```

1 retry:
2     or eax, OFFFFFFFFh
3     xbegin LO
4 LO:
5     cmp eax, OFFFFFFFFh
6     jne L1 ;jump to 'else' branch
7     ;transaction body starts here
8     ....
9     ;transaction body ends here
10    xend
11    ;transaction body ends here
12    jmp L2
13    ;actions on aborts
14 L1: ;'else' branch
15     ;actions on aborts
16     ....
17     ....
18     jmp retry
19 L2:
20     ....

```

Esempio in C con `_xbegin()` e `_xend()`

Equivalenti in assembly con `xbegin/xend`

Figura 47: Confronto tra implementazione C e assembly di una transazione TSX

6.10.0.1 Principio di funzionamento Le transazioni si appoggiano alla coerenza cache: le scritture restano nella L1 finché non si effettua il commit. Durante l'esecuzione, l'hardware tiene traccia di due insiemi:

- **Read set:** blocchi letti dalla transazione.
- **Write set:** blocchi modificati (marcati nella cache come tentativi).

Un conflitto si verifica se un altro core tenta di accedere a un blocco presente in questi insiemi; il core rileva la violazione e *aborts* la transazione, ripristinando lo stato precedente. Le istruzioni principali sono `xbegin`, `xend`, `xabort`, `xtest`. Un'implementazione comune è la **RTM (Restricted Transactional Memory)**, che richiede fallback non transazionale in caso di fallimenti ripetuti.

6.10.0.2 Casi di abort e codici di stato Quando una transazione fallisce, l'istruzione `_xbegin()` restituisce in `EAX` un **codice di stato** che indica la causa dell'abort. Le ragioni più comuni sono:

- **Conflitti di accesso:** un altro core accede (in lettura o scrittura) a una linea di cache inclusa nel *write set* della transazione corrente.
- **Interruzioni o eccezioni asincrone:** interrupt, context switch, page fault o eventi che invalidano il contesto transazionale.
- **Overflow o limiti hardware:** superamento della capacità dei buffer interni o del numero massimo di linee L1 monitorabili (tipicamente poche decine di KB).
- **Abort esplicito:** uso dell'istruzione `XABORT`, che forza la terminazione e può passare un argomento nei bit 24–31 del registro `EAX`.
- **Assenza di supporto TSX:** il bit `CPUID.07H.EBX.RTM` non è impostato, quindi le istruzioni transazionali vengono ignorate o causano eccezione.

I bit del registro `EAX` restituiti in caso di abort sono così interpretati:

EAX Bit	Significato
0	Abort causato da istruzione <code>XABORT</code>
1	Transazione potenzialmente ripetibile al retry
2	Conflitto con altro processore su indirizzo monitorato
3	Overflow del buffer interno
4	Hit di breakpoint o debug event
5	Abort durante una transazione annidata
6–23	Riservati
24–31	Argomento passato a <code>XABORT</code>

Tabella 1: Codici di stato restituiti nel registro `EAX` in caso di abort transazionale

Se nessun bit è impostato (`EAX = 0`), significa che la transazione è stata abortita senza una causa specifica riconoscibile (abort generico).

6.11 DRAM e Refresh

La **Dynamic RAM (DRAM)** conserva ciascun bit come carica elettrica in un piccolo condensatore, controllato da un transistor. Col tempo la carica si disperde attraverso resistenze parassite, quindi ogni cella deve essere periodicamente **rinfrescata** per non perdere il dato. Il tempo caratteristico di scarica è $\tau = RC$, e ogni riga viene tipicamente refreshata ogni 64 ms.

6.11.0.1 Effetti del refresh

- **Energia:** ogni ciclo di refresh consuma energia addizionale.
- **Prestazioni:** durante il refresh le righe interessate non sono accessibili.
- **Scalabilità:** più capacità significa più righe da aggiornare e quindi pause più lunghe.

Per mitigare l'impatto si adottano tecniche di **distributed refresh**: il controller rinfresca blocchi separati in tempi diversi, riducendo le pause globali. Altre strategie (es. *retention-aware refresh*) regolano dinamicamente la frequenza in base alla stabilità elettrica delle celle, riducendo consumi senza compromettere l'integrità dei dati.

Conclusione

Questo modulo mostra come l'evoluzione hardware — dal parallelismo a livello di pipeline e thread, alla gestione speculativa e coerente delle cache, fino alle transazioni e alla DRAM — influenzino direttamente i modelli di programmazione e le garanzie di sicurezza dei sistemi moderni. Ogni livello della gerarchia hardware (CPU, cache, memoria, controller) contribuisce a bilanciare potenza, latenza e coerenza, delineando le fondamenta su cui si costruiscono le architetture di sicurezza e difesa dei sistemi.