POLITECNICO DI TORINO

A.A. 2019/2020

Report 1:

Connectivity check – Advanced ping

Student:

Simone Galota - 233727 – Gruppo 16

Teacher:

Prof. Marco Mellia

Course:

Laboratorio di Internet

Configurazione di rete:

I test sono stati svolti usando la seguente configurazione a livello IP.

Indirizzo di rete	172.16.9.0/27
Indirizzo IP di H1	172.16.9.1/27
Indirizzo IP di H2	172.16.9.2/27
Indirizzo IP di H3	172.16.9.3/27
Subnet Mask	255.255.255.224
Indirizzo di broadcast	172.16.9.31

Test 1: ping agli indirizzi riservati (broadcast/rete)

1.1 Ping all'indirizzo di broadcast

In questo primo test viene eseguito un ping verso l'indirizzo di broadcast da parte dell'host H1. In Linux, usando il comando *"sysctl net.pv4.icmp_echo_ignore_broadcasts=n"*, è possibile controllare (senza "=n") se la macchina è impostata per rispondere ad un ping verso il broadcast e/o modificare il settaggio. Se n=0 l'host risponde al ping in broadcast, se n=1 l'host ignorerà il ping. Quest'ultimo accorgimento è utile per prevenire attacchi di negazione del servizio distribuiti (DDoS - Distributed Denial of Service), causati dalle risposte di tutti gli host della rete a un eccessivo numero di richieste. Le tabelle di ARP risultano vuote prima dell'inizio della sessione di ping. La sintassi del comando prevede che si aggiunga l'opzione -b per eseguire un ping all'indirizzo di broadcast.

Si pinga verso broadcast con 5 *Echo_request* da 100 bytes:

```
(base) laboratorio@laboratorio:~$ ping 172.16.9.31 -s100 -c5 -b
WARNING: pinging broadcast address
PING 172.16.9.31 (172.16.9.31) 100(128) bytes of data.
108 bytes from 172.16.9.1: icmp_seq=1 ttl=64 time=0.045 ms
108 bytes from 172.16.9.2: icmp_seq=1 ttl=64 time=1.62 ms (DUP!)
108 bytes from 172.16.9.3: icmp_seq=1 ttl=64 time=1.68 ms (DUP!)
108 bytes from 172.16.9.1: icmp_seq=2 ttl=64 time=0.058 ms
108 bytes from 172.16.9.2: icmp_seq=2 ttl=64 time=0.600 ms (DUP!)
108 bytes from 172.16.9.3: icmp_seq=2 ttl=64 time=0.769 ms (DUP!)
108 bytes from 172.16.9.1: icmp_seq=3 ttl=64 time=0.045 ms
108 bytes from 172.16.9.2: icmp_seq=3 ttl=64 time=1.37 ms (DUP!)
108 bytes from 172.16.9.3: icmp_seq=3 ttl=64 time=1.40 ms (DUP!)
108 bytes from 172.16.9.1: icmp_seq=4 ttl=64 time=5.80 ms
108 bytes from 172.16.9.2: icmp_seq=4 ttl=64 time=22.6 ms (DUP!)
108 bytes from 172.16.9.3: icmp_seq=4 ttl=64 time=22.7 ms (DUP!)
108 bytes from 172.16.9.1: icmp_seq=5 ttl=64 time=0.041 ms

--- 172.16.9.31 ping statistics ---
5 packets transmitted, 5 received, +8 duplicates, 0% packet loss, time 4035ms
rtt min/avg/max/mdev = 0.041/4.527/22.714/7.886 ms
```

figura 1.1

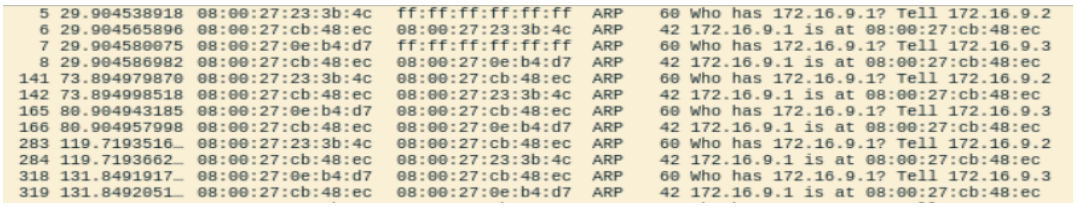
Assumiamo che, ai fini dell'esperimento, gli host siano impostati per rispondere a pacchetti inviati in broadcast. Dall'output del comando (figura 1.1) si evidenziano tre anomalie rispetto a uno scenario standard:

1) Essendo *l'Echo_request* mandata in broadcast, H1 riceverà tante *Echo_reply* quanti sono gli host all'interno della rete (compreso se stesso). Si noti come l'applicazione sia progettata **per riconoscere duplicati** (DUP!) di risposte già ricevute (cioè con lo stesso *icmp_seq*).

2) Tra le tre *Echo_reply* in risposta a ogni pacchetto inviato, la **prima ha sempre un RTT molto inferiore** rispetto alle altre due. Tale anomalia è dovuta al fatto che **la prima risposta è quella dell'host che ha pingato** verso broadcast, in questo caso H1. Per H1, sia la richiesta che la risposta sono trasmesse **sull'interfaccia virtuale di *loopback,*** per cui i tempi di invio e ricezione sono quasi immediati. Per lo stesso motivo, la risposta di H1 a se stesso non è visualizzabile durante una cattura su wireshark (sniffer che fa una copia dei pacchetti che transitano a livello 2, tra LLC e MAC). Tuttavia, se si vuole visualizzare *l'Echo_reply* in questione, basta effettuare una cattura sull'interfaccia di "Loopback: lo" oppure "any", anziché "eth0".

3) In questo caso specifico si è deciso di utilizzare anche l'opzione "-c" per limitare il numero di richieste da inviare. Possiamo osservare come la ricezione della prima delle tre risposte all'ultima richiesta (*icmp_seq=5*) faccia terminare l'applicazione, facendo sì che non ci siano risposte duplicate per quest'ultima *Echo_reply.* Se l'applicazione viene fatta partire senza "-c" si avranno tutte le risposte duplicate.

Inoltre, è stata effettuata una cattura su wireshark (figura 1.2 – filtrata per pacchetti arp) sull'interfaccia "*eth0*" di H1. Da quest'ultima si evince che non appena H2, H3 ricevono la prima *Echo_request* (inoltrata a FF:FF:FF:FF:FF:FF**, senza fare ARP-req**), essendo vuote le loro tabelle ARP, mandano in broadcast *un'ARP-request* ciascuno (righe 5-7) per conoscere l'indirizzo MAC di H1 e rispondere al ping. H1 risponde ad ognuno con *un'ARP-reply* (righe 6-8). Dopo questo step, nella tabella ARP di H1 si potranno visualizzare le corrispondenze IP/MAC di H2 e H3 (quindi due entry); invece nelle tabelle ARP di H2 e H3 sarà visualizzata solamente la corrispondenza IP/MAC di H1. Infine, possiamo notare che durante la trasmissione dei pacchetti ICMP, H2 e H3 mandano in unicast ad H1 delle ***ARP-request* di "refresh"** (ogni 40s/50s circa) per controllare se la loro tabella di ARP (che ha una sola voce) è ancora valida (righe 141 in poi).



5	29.904538918	08:00:27:23:3b:4c	ff:ff:ff:ff:ff:ff	ARP	60	Who has 172.16.9.1? Tell 172.16.9.2
6	29.904565896	08:00:27:cb:48:ec	08:00:27:23:3b:4c	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec
7	29.904580075	08:00:27:0e:b4:d7	ff:ff:ff:ff:ff:ff	ARP	60	Who has 172.16.9.1? Tell 172.16.9.3
8	29.904586982	08:00:27:cb:48:ec	08:00:27:0e:b4:d7	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec
141	73.894979870	08:00:27:23:3b:4c	08:00:27:cb:48:ec	ARP	60	Who has 172.16.9.1? Tell 172.16.9.2
142	73.894998518	08:00:27:cb:48:ec	08:00:27:23:3b:4c	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec
165	80.904943185	08:00:27:0e:b4:d7	08:00:27:cb:48:ec	ARP	60	Who has 172.16.9.1? Tell 172.16.9.3
166	80.904957998	08:00:27:cb:48:ec	08:00:27:0e:b4:d7	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec
283	119.7103516_	08:00:27:23:3b:4c	08:00:27:cb:48:ec	ARP	60	Who has 172.16.9.1? Tell 172.16.9.2
284	119.7103662_	08:00:27:cb:48:ec	08:00:27:23:3b:4c	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec
318	131.8491917_	08:00:27:0e:b4:d7	08:00:27:cb:48:ec	ARP	60	Who has 172.16.9.1? Tell 172.16.9.3
319	131.8492051_	08:00:27:cb:48:ec	08:00:27:0e:b4:d7	ARP	42	172.16.9.1 is at 08:00:27:cb:48:ec

figura 1.2

1.2 Ping all'indirizzo di rete

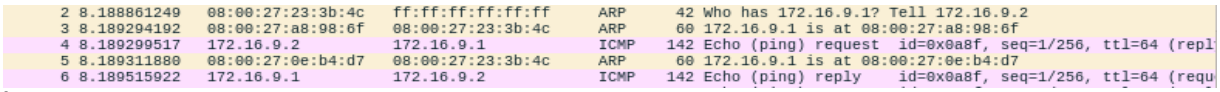
In questo test viene eseguito un ping verso l'indirizzo di rete da parte dell'host H1. Il comportamento degli host risulta analogo a quello del test precedente (1.1).

Test 2: ping con indirizzi di rete duplicati

In questo test osserviamo il comportamento della rete nel caso in cui due host vengano configurati con lo stesso indirizzo IP statico. Per cui avremo H1 e H1'(H3) con IP: 172.16.9.1/27 e H2 con indirizzo IP 172.16.9.2/27.

2.1 Ping di H2 verso l'indirizzo di H1

Eseguiamo il comando "ping 172.16.9.1 -s 100" da H2, e notiamo che il suo output sarà uguale a quello di un normale ping in unicast. Però, utilizzando wireshark sull'interfaccia *eth0* di H2, possiamo notare che nella cattura ci sono **due *ARP_reply*** (quella di H1 e di H1' – figura2.1: riga 3-5) in risposta alla normale *ARP_req* mandata in broadcast da H2 (riga 2). Sempre dalla schermata della cattura si capisce (riga 4) come H2 tenga in considerazione **solo la prima ARP_reply ricevuta**, mentre la seconda viene scartata (contromisura **all'ARP spoofing**? Sì. Se ho una richiesta, accetto solo una risposta. E se l'attaccante è nel pacchetto che ho accettato? Non lo possiamo sapere, non esiste sicurezza al 100%.). Tale scelta è non-deterministica: potrebbe toccare sia ad H1 che ad H1'. Inoltre, c'è da dire che **wireshark rileva l'esistenza di un indirizzo IP duplicato**.



2	8.188861249	08:00:27:23:3b:4c	ff:ff:ff:ff:ff:ff	ARP	42	Who has 172.16.9.1? Tell 172.16.9.2
3	8.189294192	08:00:27:a8:98:6f	08:00:27:23:3b:4c	ARP	60	172.16.9.1 is at 08:00:27:a8:98:6f
4	8.189299517	172.16.9.2	172.16.9.1	ICMP	142	Echo (ping) request id=0x0a8f, seq=1/256, ttl=64 (repl
5	8.189311880	08:00:27:0e:b4:d7	08:00:27:23:3b:4c	ARP	60	172.16.9.1 is at 08:00:27:0e:b4:d7
6	8.189515922	172.16.9.1	172.16.9.2	ICMP	142	Echo (ping) reply id=0x0a8f, seq=1/256, ttl=64 (requ

figura 2.1

Dopo aver scelto l'host, l'applicazione scorre normalmente (avremo, al solito, scambio di pacchetti ARP di *refresh* tra gli host che si stanno pingando). Infatti, se osserviamo **la cattura dal punto di vista dell'host H1 e/o H1'**, notiamo come in una delle due ci sia **traccia solamente di due pacchetti** (ARP_req di H2 e la rispettiva ARP_reply scartata da H2), mentre **nell'altra si trovano sia i due pacchetti del protocollo ARP che quelli ICMP** dell'applicazione ping. Tuttavia, visualizzando le tabelle ARP, avremo che: quella di H2 conterrà la corrispondenza dell'IP pingato con l'indirizzo MAC dell'host che ha risposto per primo; invece quelle di H1 e H1' conterranno entrambe il match IP(H2) / MAC(H2).

2.2 Ping di H1 e H1' verso l'indirizzo di H2

Eseguiamo il ping da H1 verso H2. Appena H1 inizia a pingare, avviamo il ping da H1' a H2. Vediamo cosa succede.

Sia H1 che H1' generano *un'ARP_req* in broadcast per conoscere l'indirizzo MAC di H2. Avendo avviato prima H1, la sua *ARP_req* arriva prima e quindi anche *l'ARP_reply*. H1 riempie la sua tabella di ARP con il match IP(H2) / MAC (H2) e **H2 riempie la sua ARP-table con l'IP (H1) e il MAC (H1)**. Subito dopo, *l'ARP_req* di H1' arriva ad H2 che, rispondendo con *un'ARP_reply*, **aggiorna la sua tabella di ARP con lo stesso IP di prima** (ip di H1 = ip di H1') **e l'indirizzo MAC di H1'.** H1' riempirà la sua tabella con i dati di H2. Dopo 5s dall'assegnazione, H2 controlla in unicast se è ancora valida l'informazione IP/MAC di H1' con *un'ARP_request* di refresh (fig. 2.2 – riga 27).

E' possibile vedere dalla figura 2.2 riportata sotto, come per ogni *icmp_seq* arrivino due *Echo_request* (una da H1 e una da H1') a cui corrispondono due *Echo_reply* di H2, le quali arrivano allo stesso host; cioè l'host il cui indirizzo MAC è associato all'IP sorgente [172.16.9.1] nella tabella di ARP di H2 (al momento delle richieste – nel caso in questione l'host è H1').

Dunque, H1 continua a mandare richieste senza avere risposte, per cui dopo 20s/30s H1 invia un'ARP_req unicast ad H2 per controllare se ciò che lui ha in arp-table sia ancora valido. Quando H2 la riceve, risponde con un'ARP_reply ad H1 e aggiorna la sua tabella di ARP con il MAC address di H1, il quale ricomincia a ricevere Echo_reply (sue e di H1'), mentre H1' non ottiene più risposte alle sue *Echo_request*. Ma la tabella di ARP di H2 si riaggiornerà un'altra volta, non appena H1' farà la stessa cosa di H1, ovvero *ARP_request* in unicast. **Così le *echo_reply* si alterneranno tra H1 e H1' ogniqualvolta H2 riceva una *ARP_request* in unicast.**

Quindi, ogni host che riceve risposta ne riceve due per ogni richiesta. Mentre le tabelle di ARP di H1 e H1' rimangono inalterate per tutta la durata dell'applicazione, quella di H2 verrà modificata come mostrato in figura 2.3.

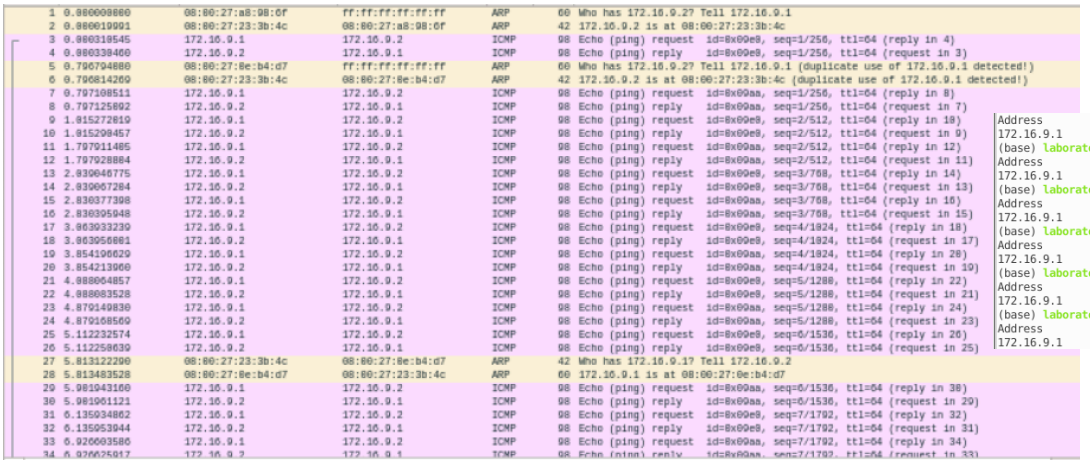


figura 2.2: cattura wireshark

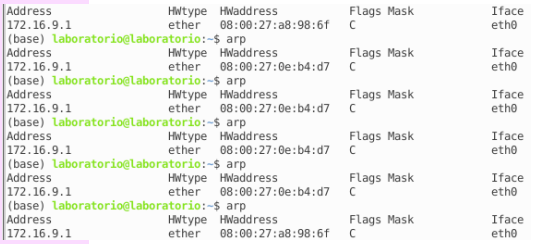


figura 2.3

Test 3: ping con configurazioni maschere errate

Configuriamo H1 in modo che veda H2 appartenente alla sua sottorete, e H2 in modo che non veda H1 appartenente alla sua. Realizziamo questa configurazione nel modo seguente:

IP address H1	172.16.9.111/24 (Subnet Mask: 255.255.255.0)
IP address H2	172.16.9.25/27 (Subnet Mask: 255.255.255.224)

Quindi, la sottorete di H1 ha $2^8 - 2 = 254$ possibili host, invece quella di H2 ha $2^5-2 = 30$ possibili host.

3.1 Ping di H1 ad H2

Eseguo da H1: "ping 172.16.9.25 -s 100 -c 10". L'output del test per ogni pacchetto è "Host destinazione irraggiungibile". Dalla cattura dei pacchetti (**fig. 3.1**) notiamo che H1 invia ripetutamente (a gruppi di 3) *ARP_req* in broadcast, **senza ricevere riscontri da parte di H2 nonostante riceva le richieste** sia a livello fisico che a livello 2. Perché appena H2 riceve *un'ARP_request*, si hanno 3 approcci possibili (a seconda del sistema operativo):

1) Si riconosce H1 come non appartenente alla stessa sottorete, quindi H2 non risponde. Questa è la scelta (più sicura) adottata su Linux, anche se comporta una violazione dei principi di separazione delle funzionalità, perché ARP deve conoscere i concetti di sottorete e indirizzi che dovrebbero essere problemi di routing e quindi di IP;

2) essendoci il collegamento fisico, H2 risponde all'ARP_req. Stabilendo così un collegamento logico punto-punto. Avendo aperto il canale, arriverebbe la *Echo_request* al livello 4, ma al momento della *Echo_reply*, a livello IP, ci sarebbe un problema di instradamento (come nel test 3.2). Inoltre, potrebbe essere pericoloso, guardando l'aspetto sicurezza, perché: se anziché avere un messaggio al secondo, avessimo migliaia di *Echo_request* al secondo alle quali rispondere (senza poterlo fare), si intaserebbe il canale creato, oltre a mandare in down l'host che le riceve.

3) L'approccio più complesso ed efficace è rispondere all'*ARP_Req* e aggiungere una route a livello 3, una "host specific route". Questo vuol dire che ARP deve anche saper modificare le tabelle di routing.

Per cui nel nostro caso (1) **la tabella di ARP di H1 contiene una entry incompleta**, quella di H2 è completamente vuota.

3.2 Ping di H2 ad H1

Facendo ping da H2 verso H1, il messaggio che da l'applicazione ping è "network is unreachable", perché c'è un problema al livello dell'instradamento. H2 non è in grado di inviare pacchetti ad H1 dato che non esiste una route specifica per raggiungere H1. Basti pensare al fatto che l'indirizzo di broadcast (che è l'ultimo indirizzo che sa raggiungere) della sottorete di H2 è 172.16.9.31, mentre l'indirizzo IP destinazione di H1 è 172.16.9.111.

In questo caso la **cattura con wireshark risulta vuota**, perché il pacchetto ICMP partito da H2 si è bloccato al livello IP. Non è possibile inviare il pacchetto per consegna diretta (0.0.0.0), poiché al **livello 3 fallisce il controllo di appartenenza** dell'host destinazione alla stessa sottorete di H2 e, non essendo impostato nessun default gateway, la rete destinazione risulta irraggiungibile. **In questo caso le tabelle di ARP di H1 e H2 rimangono vuote**.

1	0.000000000	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
2	1.027581388	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
3	2.051326087	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
4	3.075791344	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
5	4.100021698	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
6	5.123382991	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
7	6.148136818	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
8	7.172019179	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
9	8.196240815	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
10	9.220508519	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
11	10.243325496	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111
12	11.267663575	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.111

figura 3.1:

2	87.4115…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=1/256, ttl=64 (no response found!)
3	87.4115…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
4	88.4154…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
5	88.4265…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=2/512, ttl=64 (no response found!)
6	89.4398…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
7	89.4503…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=3/768, ttl=64 (no response found!)
8	90.4745…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=4/1024, ttl=64 (no response found!)
9	90.4746…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
11	91.4881…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
12	91.4983…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=5/1280, ttl=64 (no response found!)
13	92.5119…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
14	92.5230…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=6/1536, ttl=64 (no response found!)
15	93.5463…	172.16.9.25	172.16.9.31	ICMP	98 Echo (ping) request id=0x085f, seq=7/1792, ttl=64 (no response found!)
16	93.5464…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
17	94.5600…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31
18	95.5833…	08:00:27:a8:98:6f	ff:ff:ff:ff:ff:ff	ARP	42 Who has 172.16.9.25? Tell 172.16.9.31

figura 4.1:

Test 4: ping configurazioni maschere errate (H1 è indirizzo di broadcast per H2)

In questo caso la configurazione sarà la seguente:

IP address H1 (indirizzo broadcast per sottorete di H2)	172.16.9.31/24 (Subnet Mask: 255.255.255.0)
IP address H2	172.16.9.25/27 (Subnet Mask: 255.255.255.224)

4.1 Ping di H1 ad H2

L'output è nuovamente "Destination Host Unreachable". **Le tabelle di ARP** si trovano nella stessa situazione del punto 3.1. Dalle catture dei pacchetti di H1 e H2, si nota che gli unici trasmessi sono le *ARP_request* di H1 inviate in broadcast per trovare l'indirizzo MAC di H2. La richiesta non avrà una risposta da parte di H2 **perché a chiederla è un indirizzo IP che corrisponde all'indirizzo di broadcast della sua sottorete**. Si ha così una **violazione della semantica del protocollo**, perché non si può accettare che un indirizzo di broadcast diventi unicast.

4.2 Ping di H2 ad H1

Pingando da H2 a H1 bisogna aggiungere l'opzione -b come nel test 1.1, poiché per H2 l'indirizzo IP di H1 è il broadcast. L'output dell'applicazione mostra come il comando "ping" vada a buon fine, anche se **l'unica risposta che H2 riceve per ogni pacchetto inviato è quella della sua interfaccia virtuale di loopback.**
Prestando attenzione alla traccia (**figura 4.1 - sopra**) di wireshark su H1 o H2 (su interfaccia eth0 sostanzialmente è uguale), si può notare come nessuna *Echo_reply* è trasmessa da H1 ad H2. Infatti, sono solo presenti *Echo_request* (trasmesse in broadcast) e *ARP_request* di H1 in broadcast (per conoscere il MAC di H2) così da poter riempire la sua tabella di ARP e inviargli una *Echo_reply*. Però non si osserva **nessuna *ARP_reply* da H2 ad H1, dato che a chiedergliela è il suo indirizzo di broadcast** (violazione semantica protocollo).
Le tabelle di ARP non variano: H1 avrà una entry incompleta e H2 sarà vuota poiché non ha mai avuto bisogno di conoscere l'indirizzo MAC di H1.