

OVS と Linux ネットワークで 実現する HV の透過的プロキシ

近藤智文 / サイバーエージェント
CODT2026

近藤 智文

- サイバーエージェント24新卒
- プライベートクラウド IaaS 基盤の開発・運用をやっています
- OpenStack/OVS/Kubernetes /Go/Python



<https://github.com/TOMOFUMI-KONDO/>

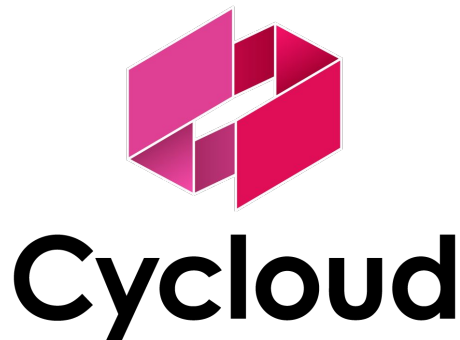
https://x.com/tomokon_0314

- 高いテナント分離性と undercloud の疎通性を両立する仮想ネットワーク設計
- 上記を構成する OVS や Linux ネットワーク機能の技術的詳細

プライベートクラウド 新リージョンの VPC 要件

社内ユーザーに以下の機能を提供

- IaaS (OpenStackベース)
- PaaS/SaaS
 - Kubernetes as a Service
 - Serverless Computing
 - ML Platform
 - GitHub Actions Self-Hosted Runner
 - DB/Storage, etc.

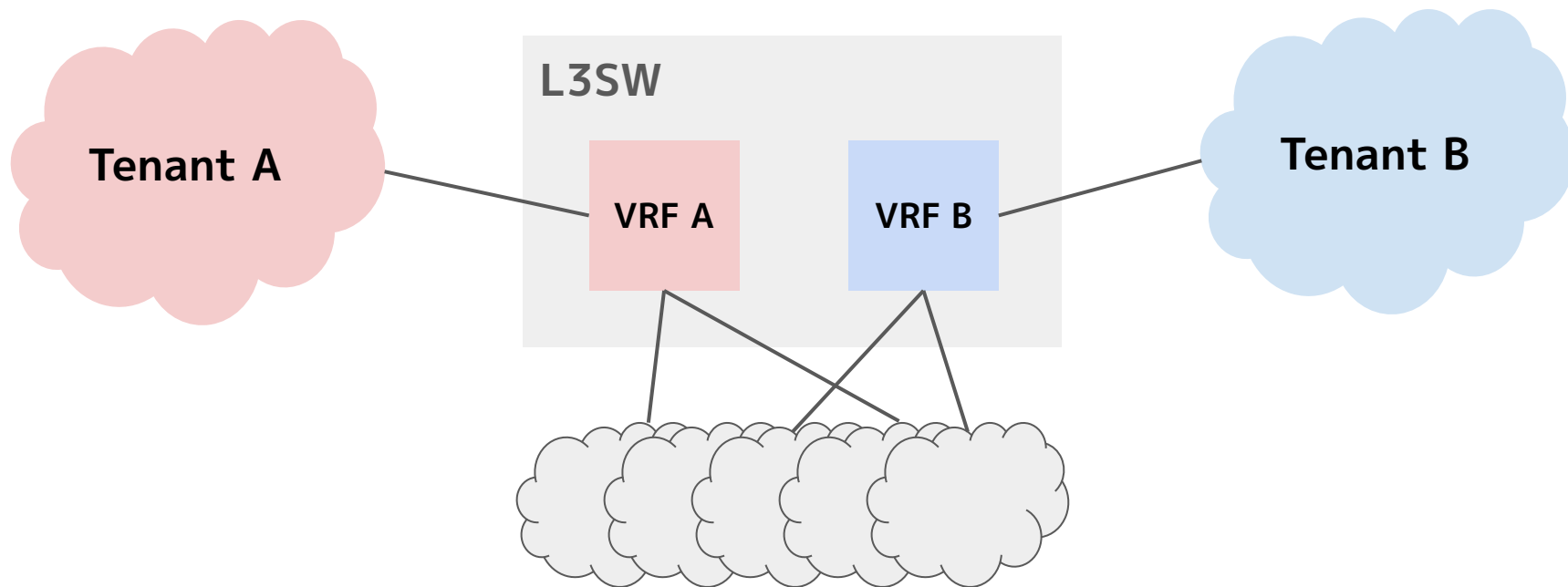


ハードもソフトもゼロベースで構築

- 最新世代のハードウェアによりリソース効率やパフォーマンスの大幅向上
- 統合されたUI/UX・メンテナンスタイムの削減による利用者の負担削減
- 高い**テナント分離性**によるセキュリティ向上←←←

テナント分離とは？

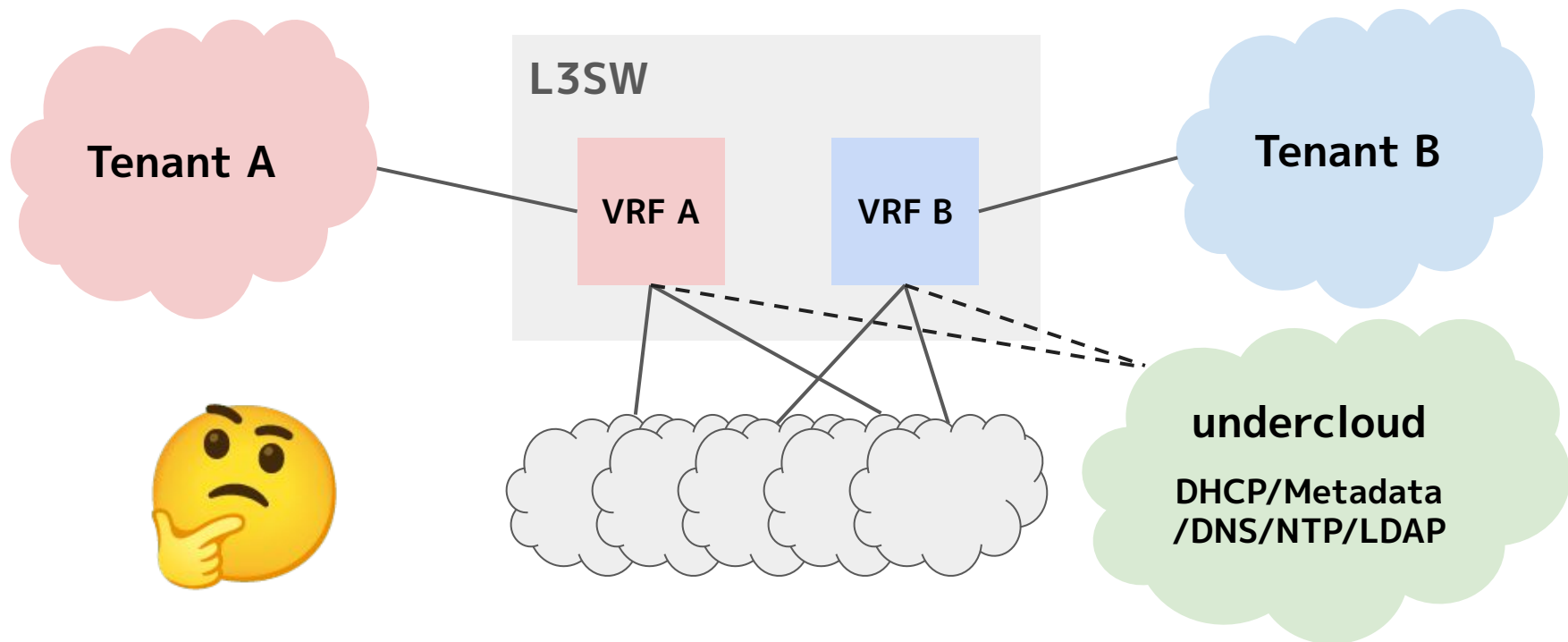
各テナントの VPC をL3レベルで分離



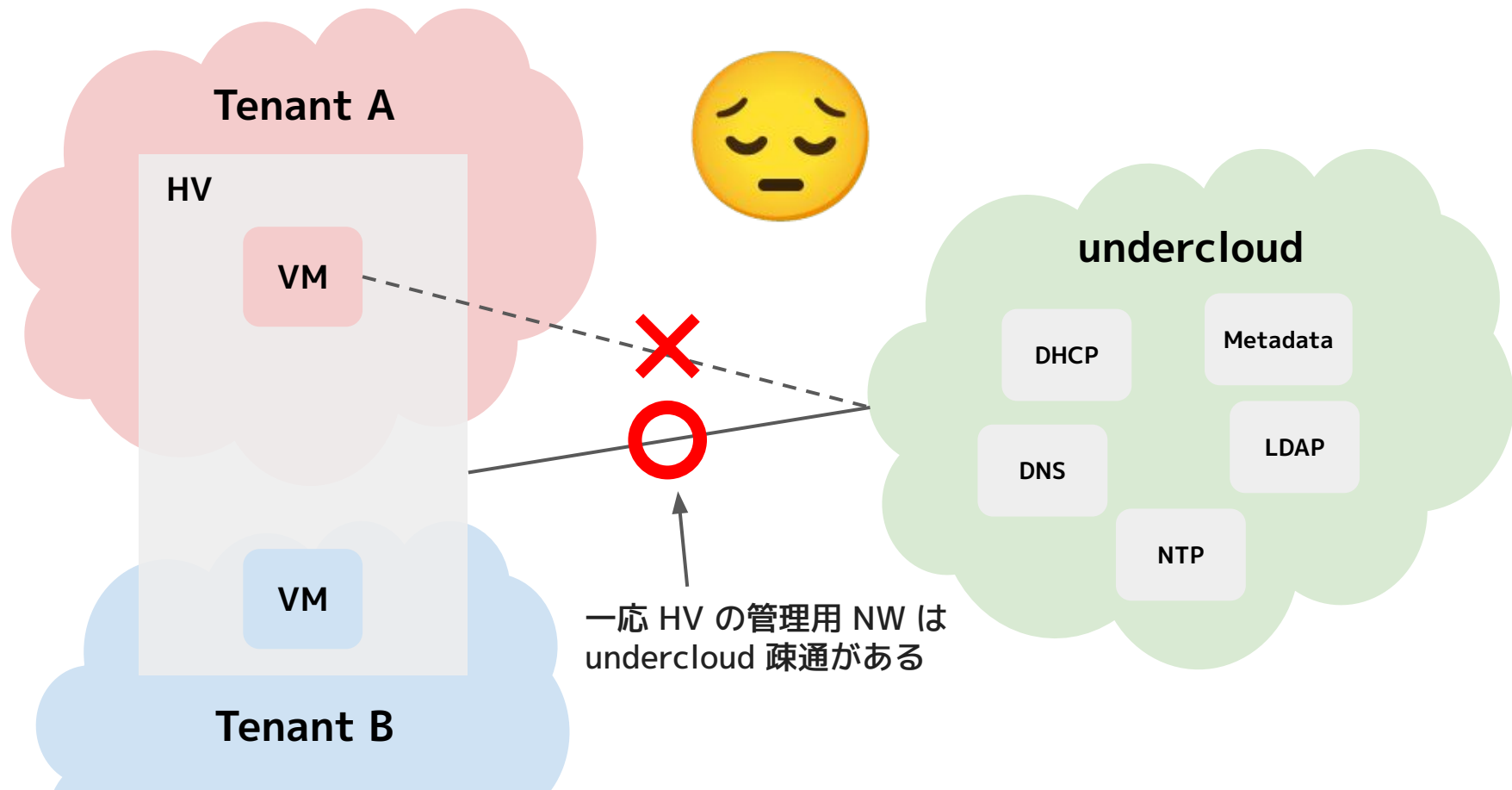
セキュアでいいじゃん！

しかし...

各テナントの VPC を L3 レベルで分離



VM が undercloud のサービスにアクセスできない



Metadata が取れないので
cloud-init が動かない！

undercloud 疎通がある

Tenant B

というかそもそも DHCP が
返ってこないなのでアドレスが
振られない 😊

undercloud 疎通がある

Tenant B

undercloud を全テナントに疎通させればいいのでは？

一見良さそう。だが...

- 逆に言えば、undercloud に侵入されると**全てのテナントに疎通できてしまう**
- せっかくテナント分離をしているのにそのような大きな侵入口を作ってしまうのは避けたい

矛盾する二つの要件を同時に満たす方法はあるのか...

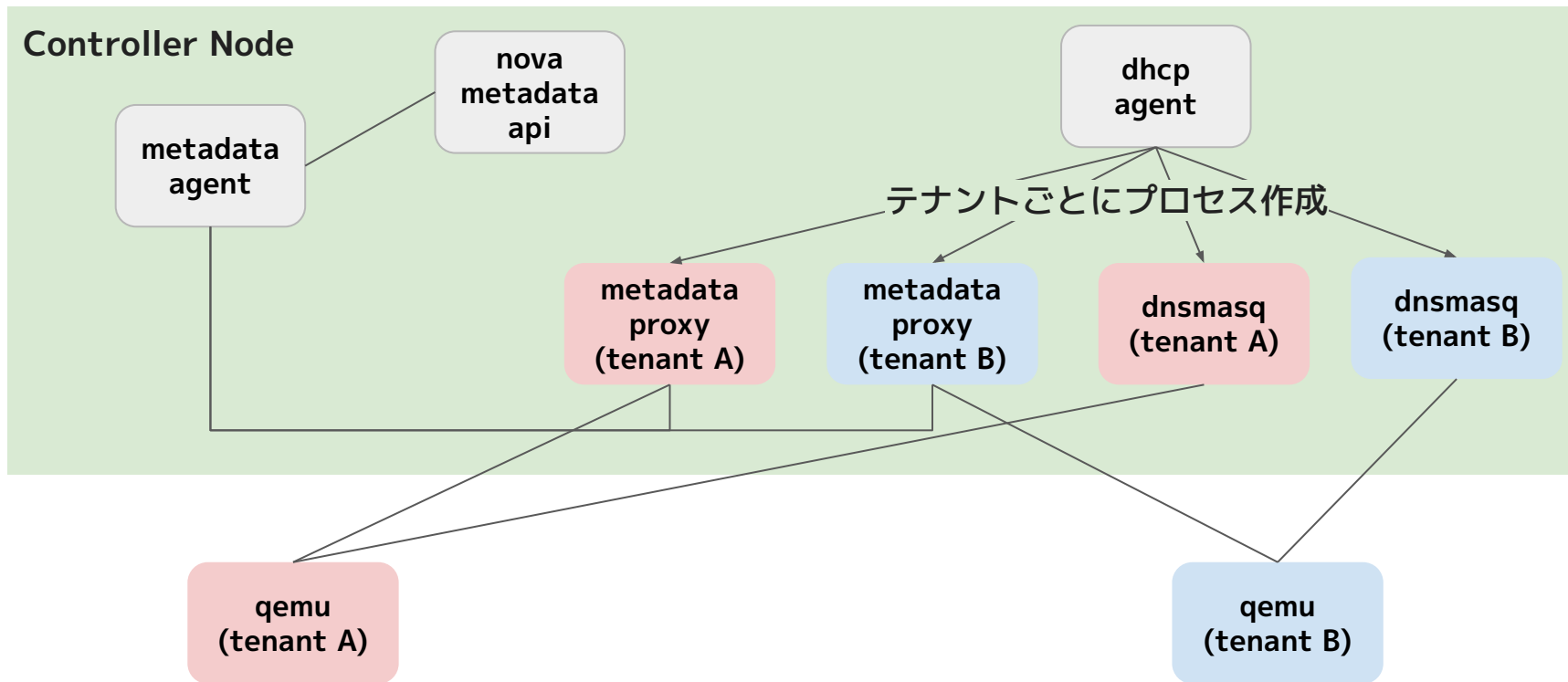
- セキュリティの観点から各テナントは NW 的に完全に分離したい
- VM から undercloud のサービスに疎通させたい



Neutron の DHCP/Metadata おさらい

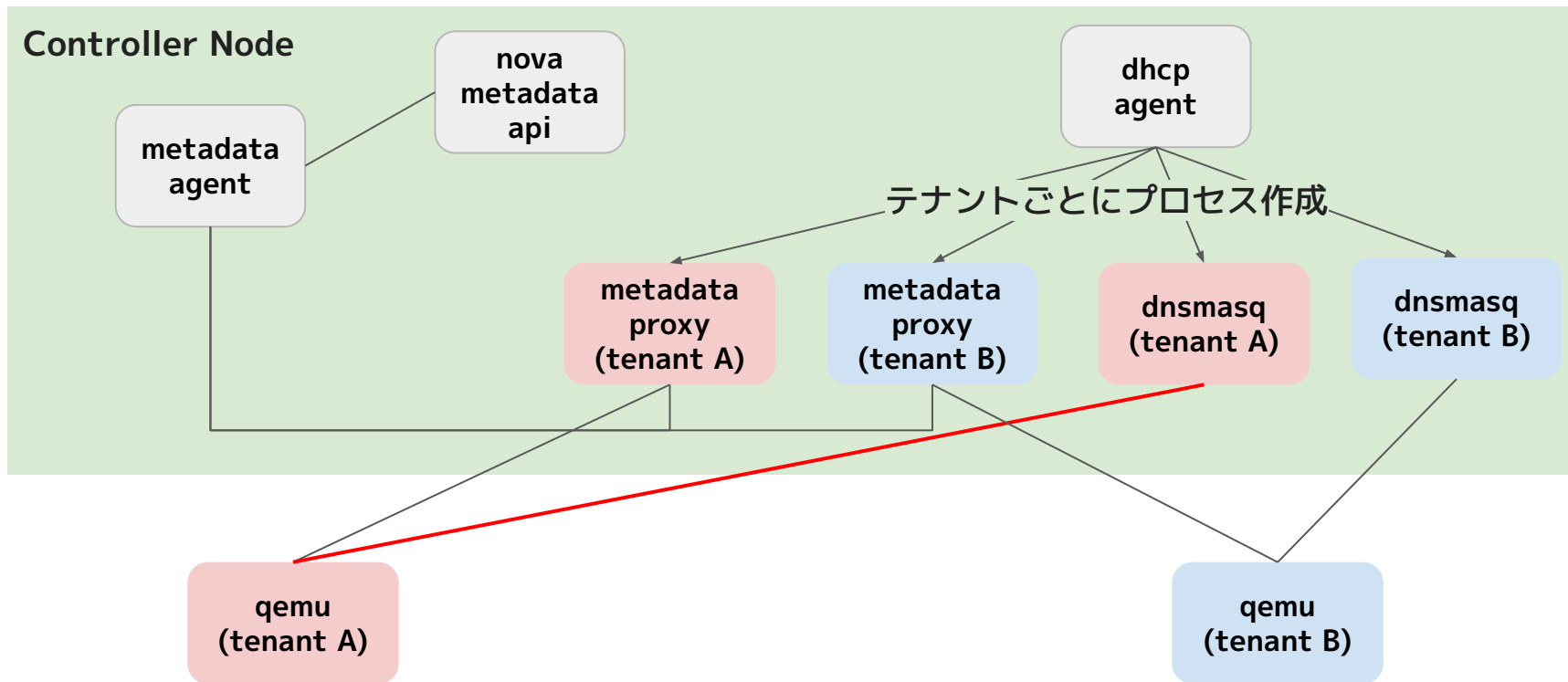


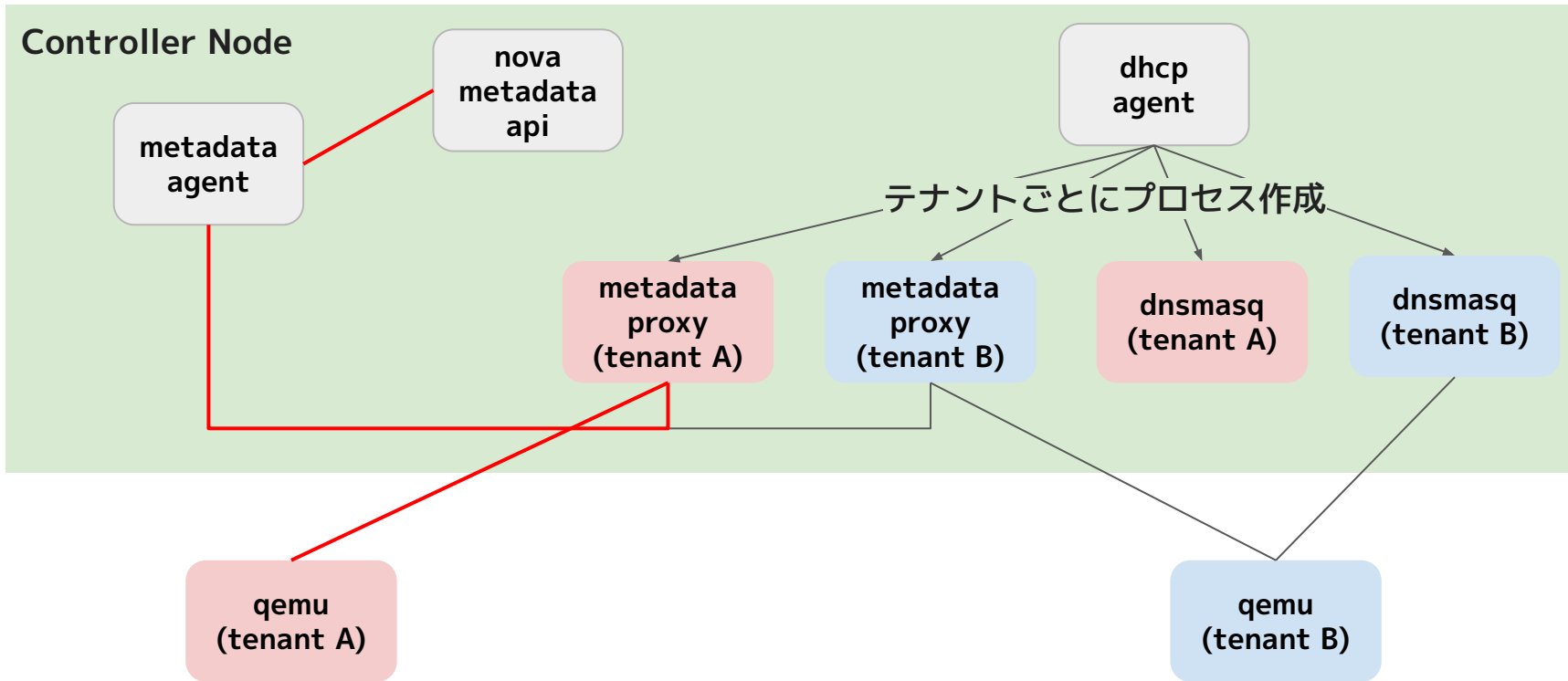
Neutron の DHCP/Metadata 構成



(Neutron: OpenStack のネットワークサービス)

DHCP Neutron の DHCP/Metadata 構成





Controller Node

nova

metadata
agent

Control Plane が
全テナントに疎通する
必要がある！

dnsmasq
(tenant B)

qemu
(tenant A)

qemu
(tenant B)



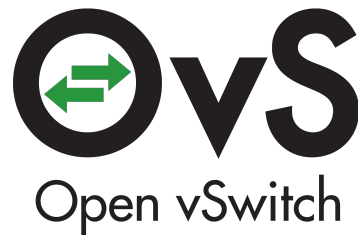
- agent を集約すると Control Plane が**全テナントに L2 接続する必要**がある ✖
 - 別途 Network Node を用意したりするのは運用コスト的に厳しい
- DHCP/Metadata 以外の通信に対応できない ✖
 - DNS, NTP, LDAP, etc.



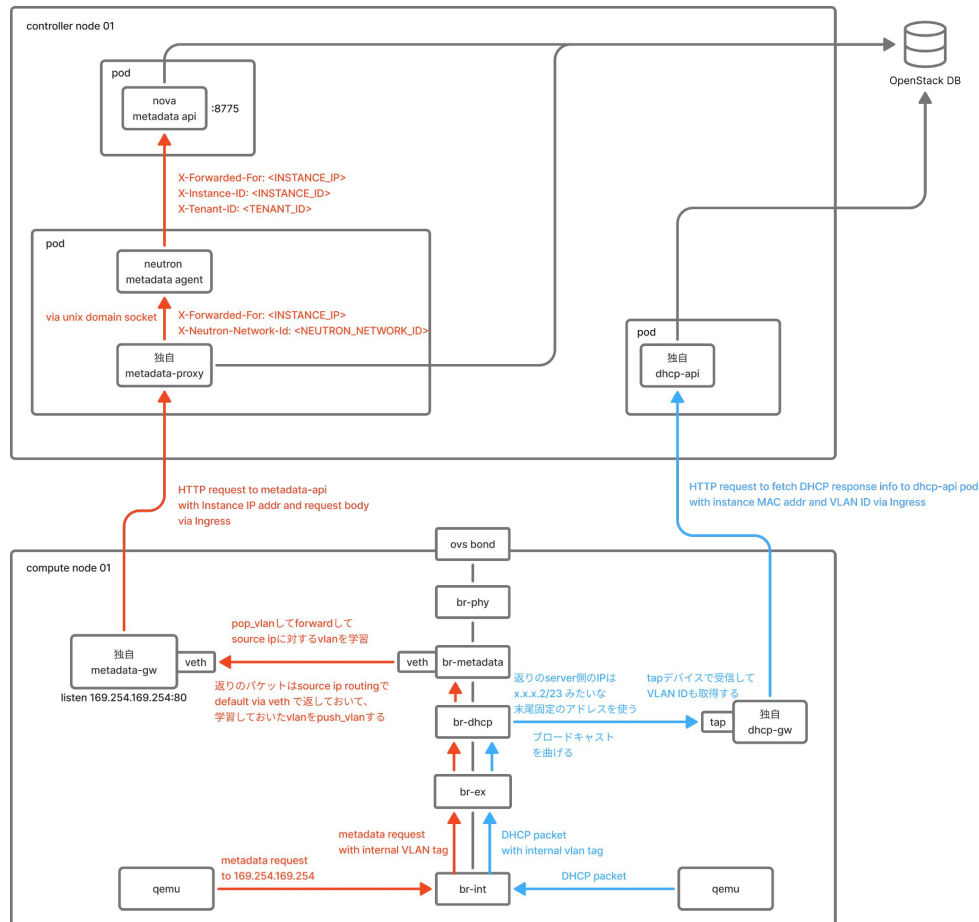
でも大丈夫

Open vSwitch

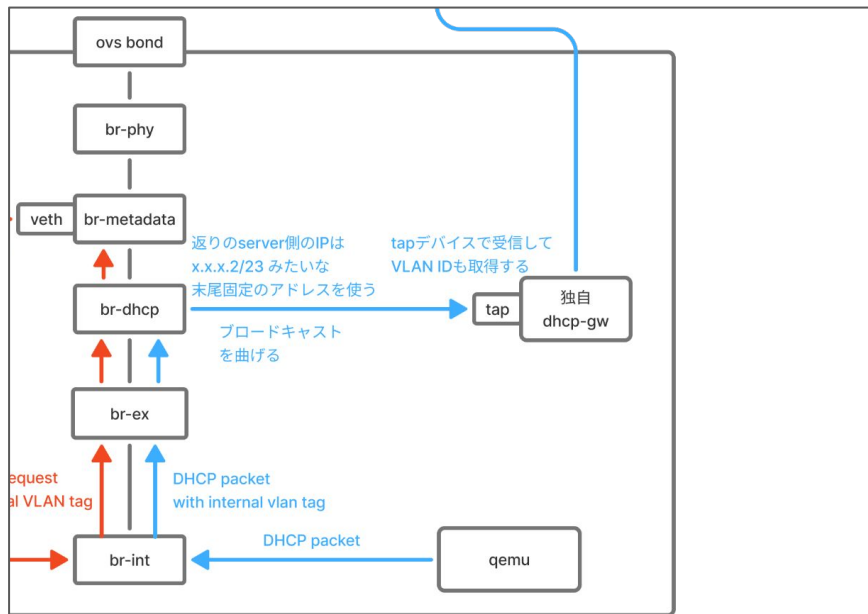
- Linux で動作する仮想スイッチ OSS
- Neutron のデータプレーンとして既に導入済みだった
- 通過するパケットに対して任意のフォーワーディングや中身の書き換えが可能



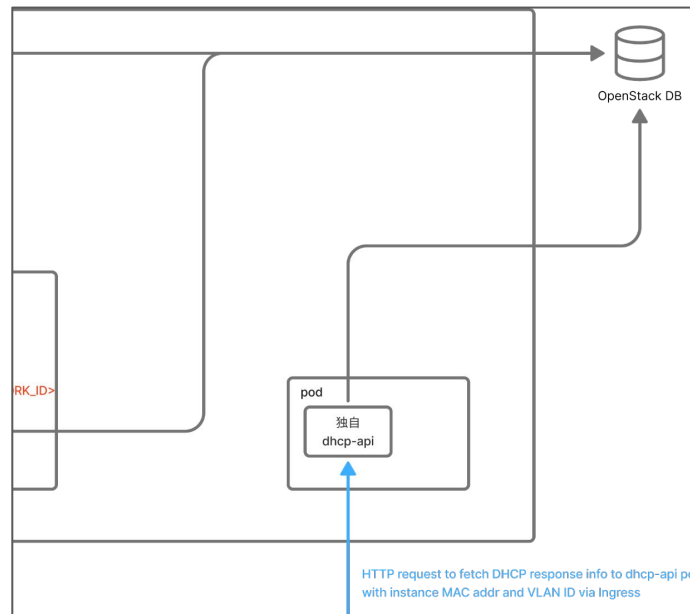
設計 全体像



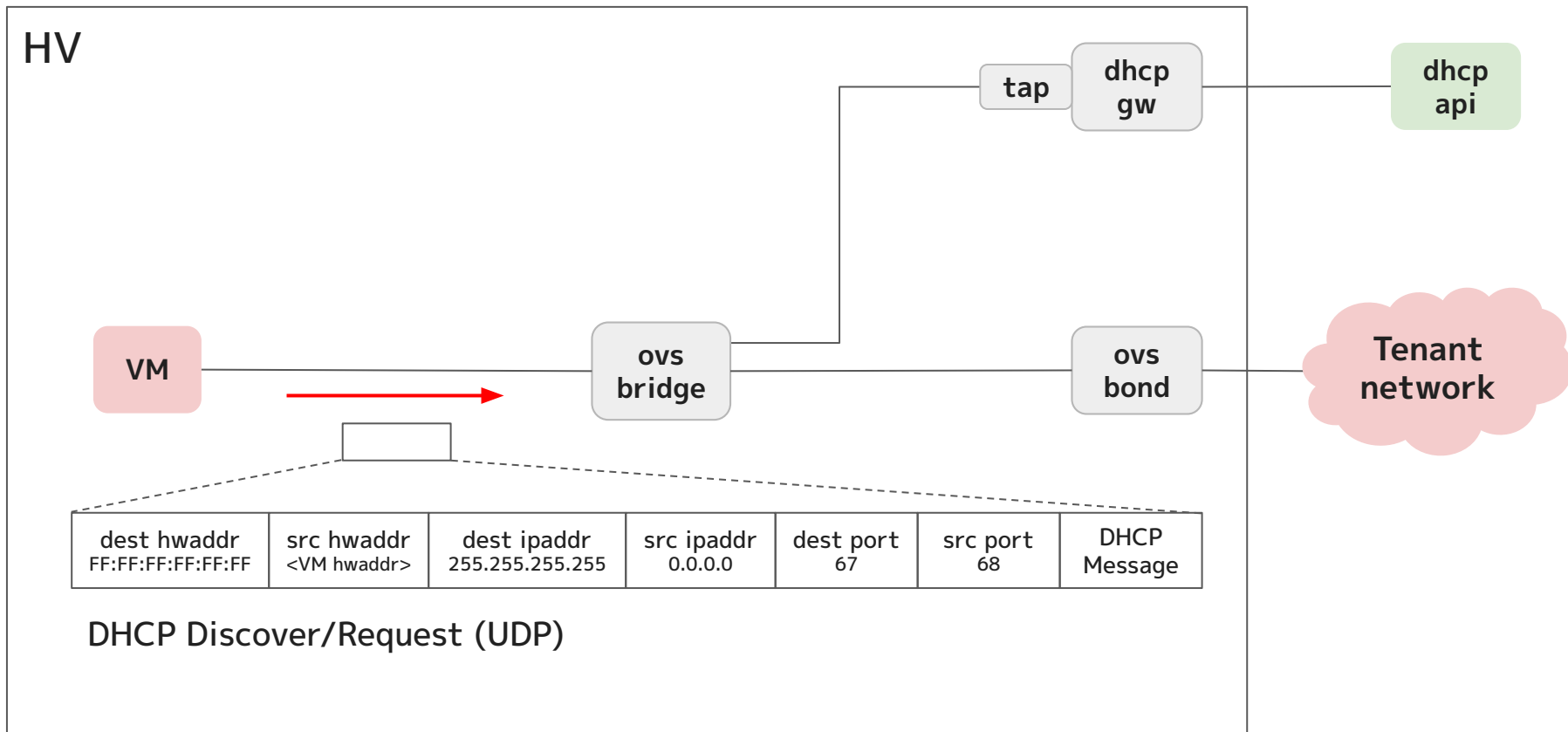
HV



Controller



DHCP OVS 詳細 (行き)



DHCP OVS 詳細 (行き)

HV

in_port=<VM port>,
udp,udp_src=68,udp_dst=67,
nw_dst=255.255.255.255,
actions=output:<dhcp-gw tap>

VM

ovs
bridge

tap

dhcp
gw

dhcp
api

DHCP のみ
tap へ曲げる

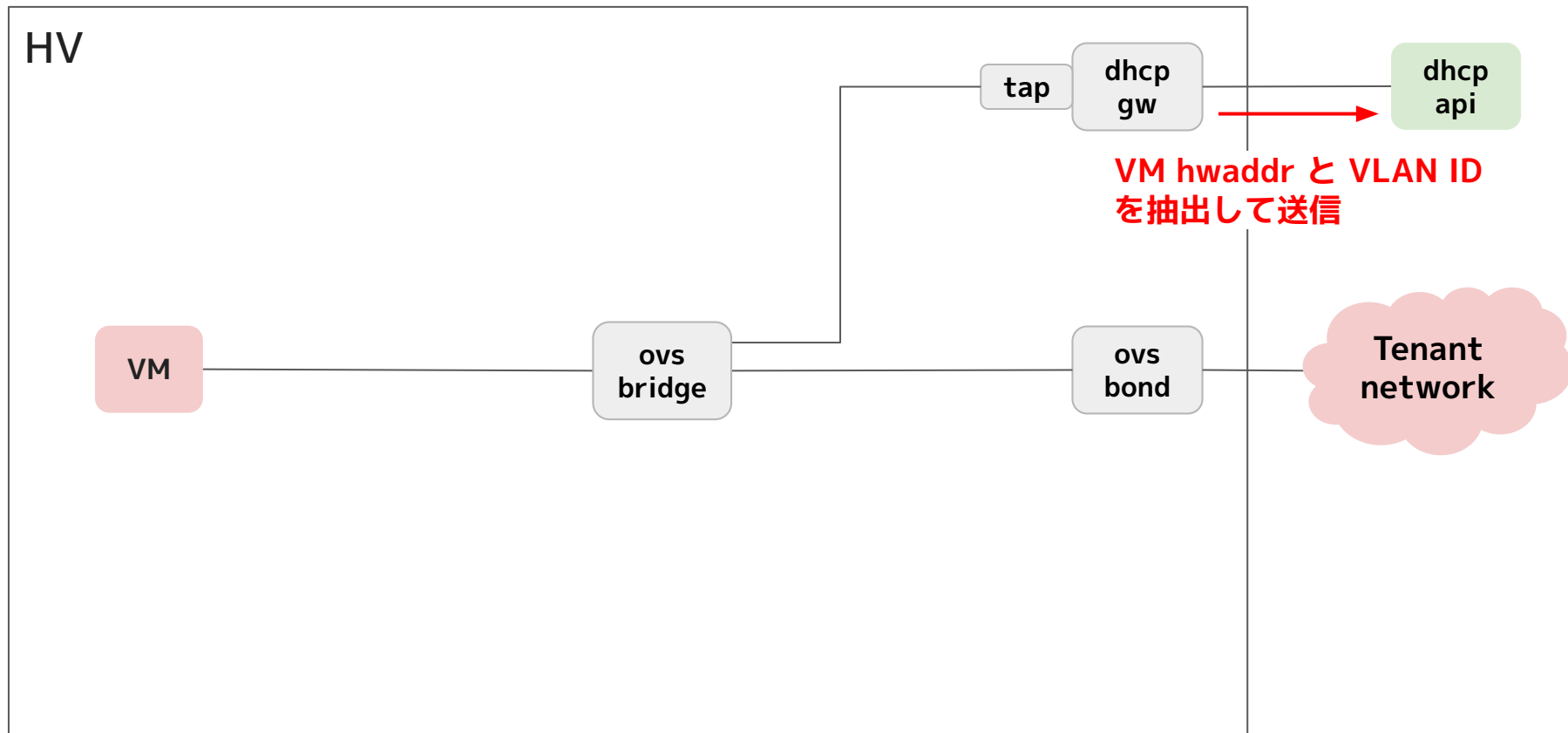
ovs
bond

Tenant
network

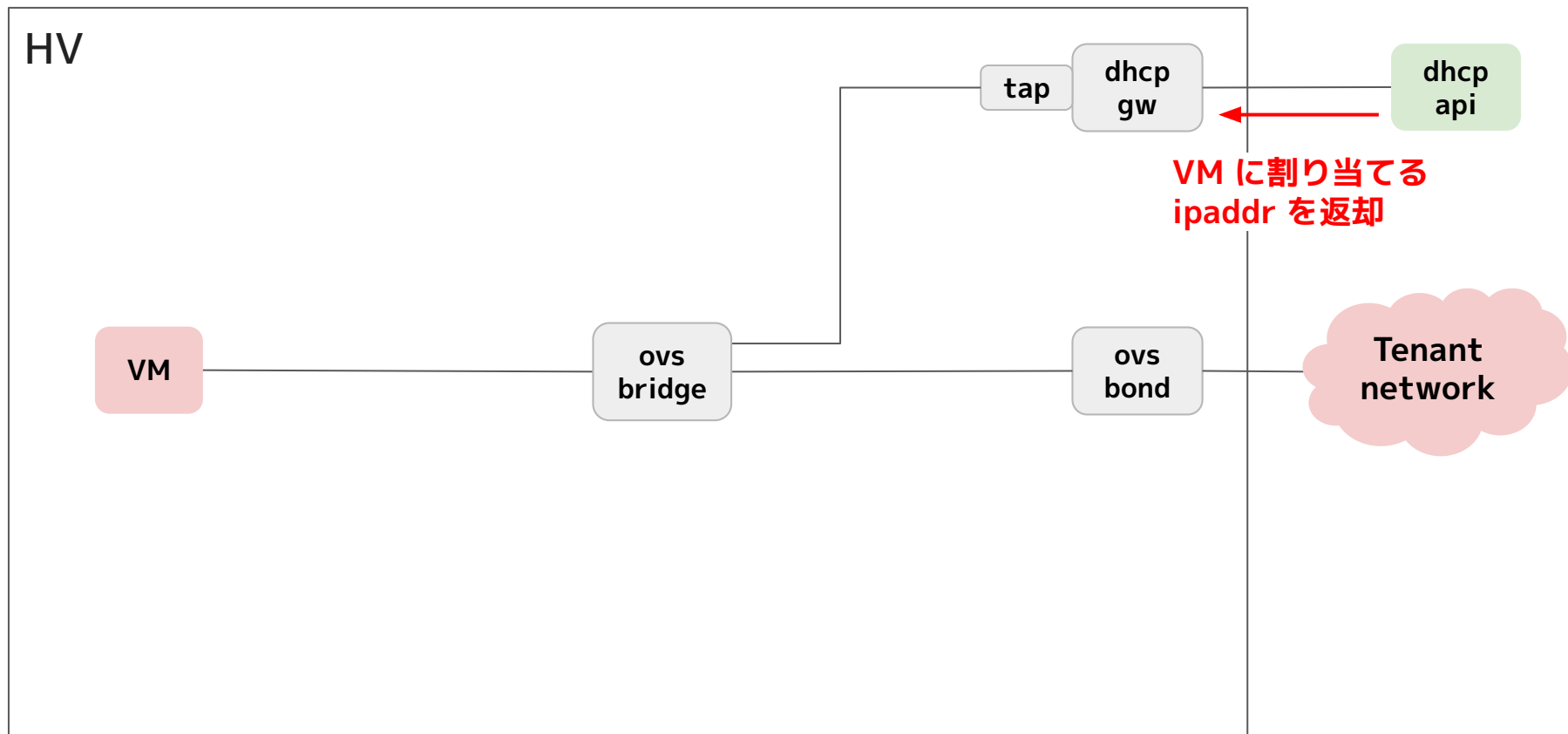
dest hwaddr FF:FF:FF:FF:FF:FF	src hwaddr <VM hwaddr>	dest ipaddr 255.255.255.255	src ipaddr 0.0.0.0	dest port 67	src port 68	DHCP Message
----------------------------------	---------------------------	--------------------------------	-----------------------	-----------------	----------------	-----------------

DHCP Discover/Request (UDP)

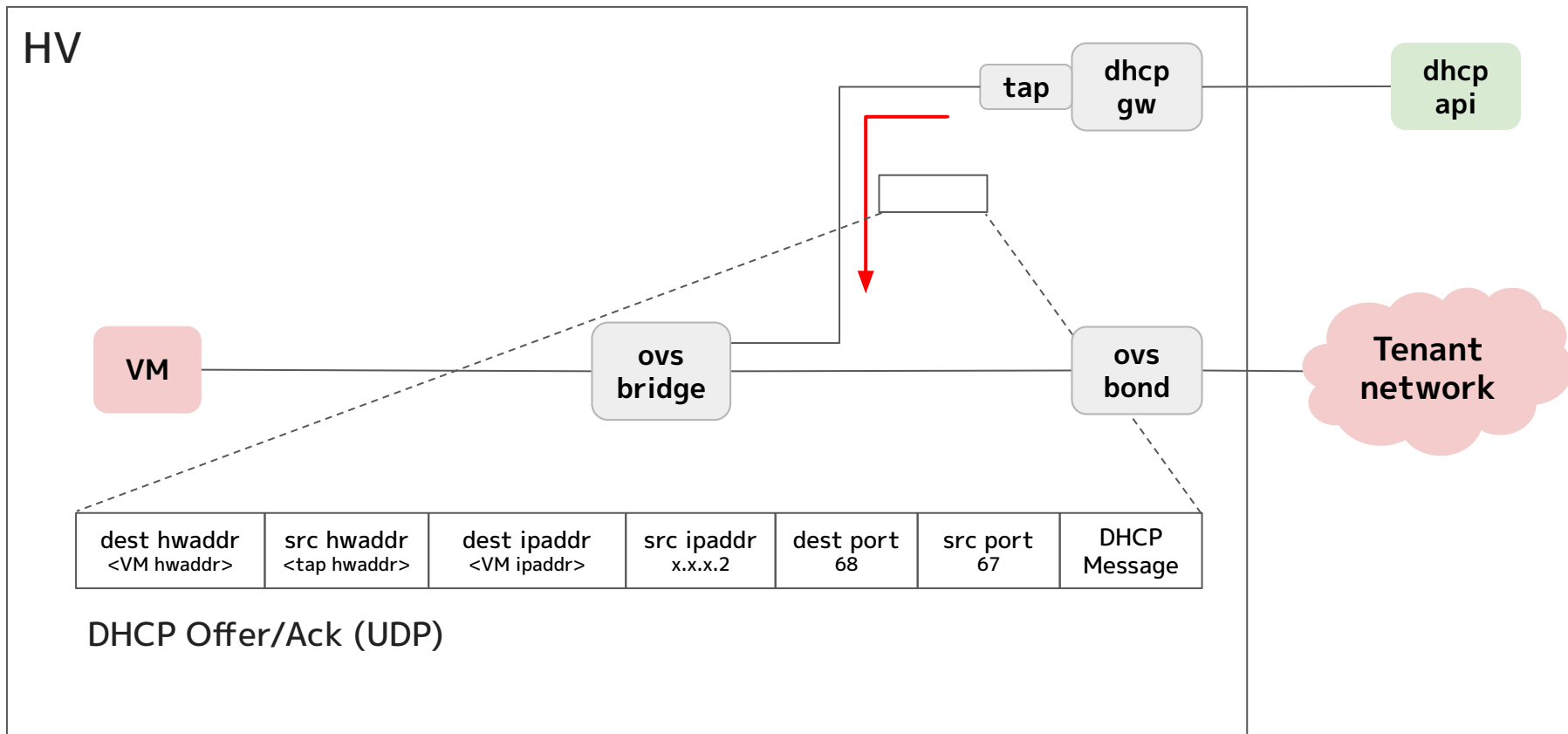
DHCP OVS 詳細（行き）



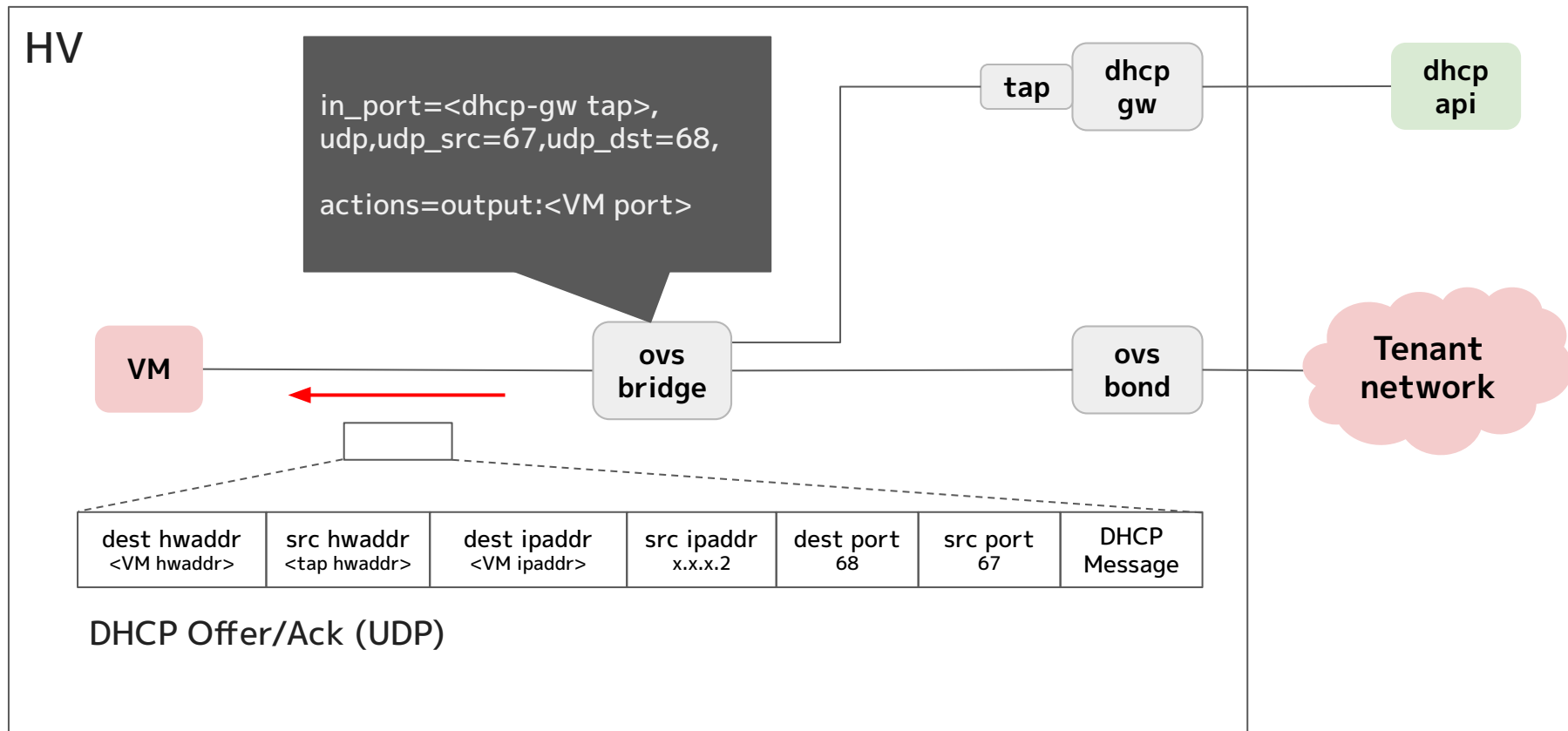
DHCP OVS 詳細 (帰り)



DHCP OVS 詳細 (帰り)

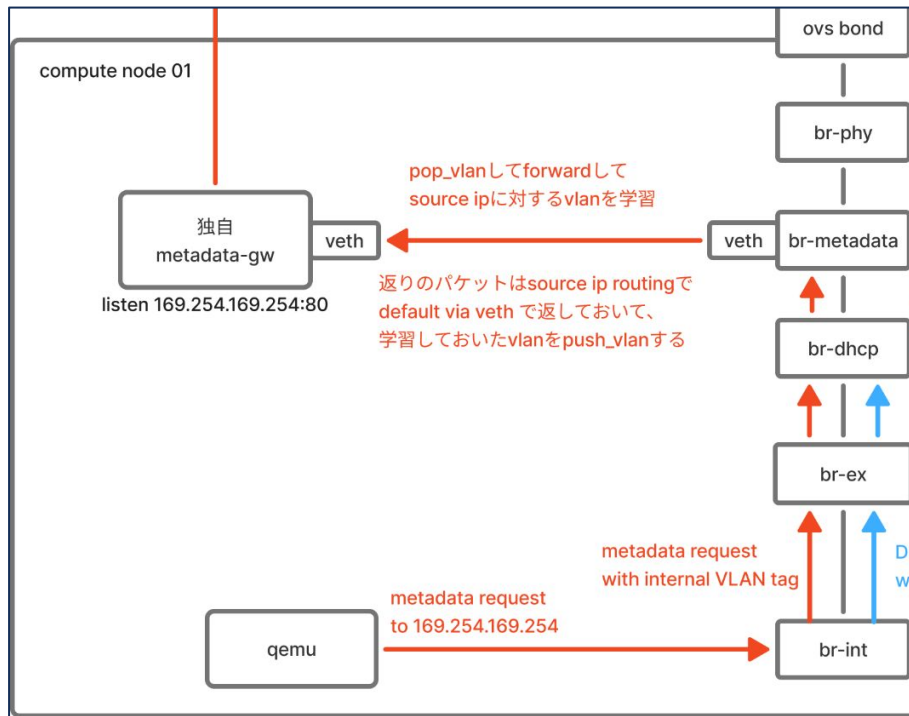


DHCP OVS 詳細 (帰り)

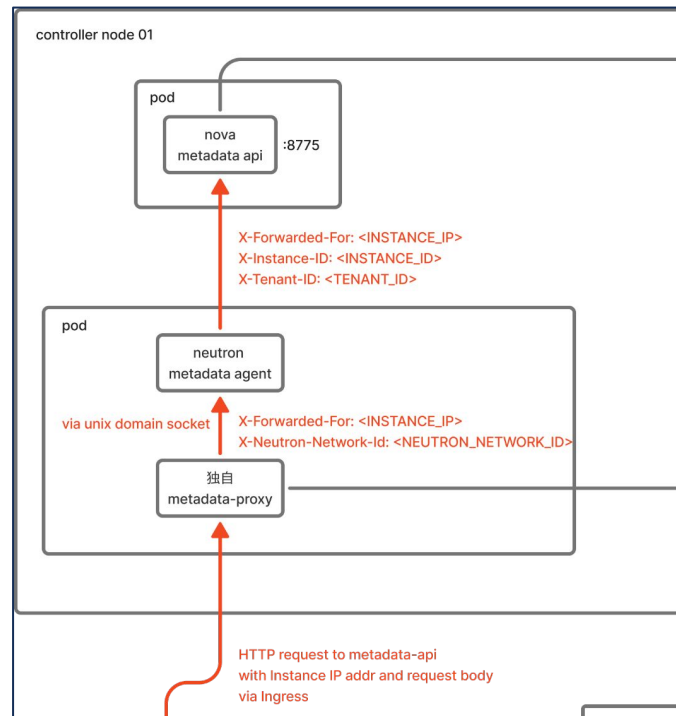


設計 Metadata (DNS / NTP / LDAP もほぼ同様)

HV



Controller



Metadata OVS 詳細 (ARP / 行き)

HV

```
$ ip route show
```

```
169.254.169.254 dev enp1s0  
proto dhcp scope link
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	arp op 1	sender hwaddr <VM hwaddr>	sender ipaddr <VM ipaddr>	target hwaddr 00:00:00:00:00:00	target ipaddr 169.254.169.254
-----------------------------	-------------	------------------------------	------------------------------	------------------------------------	----------------------------------

ARP Request

Metadata OVS 詳細 (ARP / 行き)

HV

```
table=0,in_port=<VM port>,  
vlan_tci=0x1000/0x1000,  
arp,arp_tpa=169.254.169.254,arp_op=1,  
actions=resubmit(,20)
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

169.254.169.254
当での ARP Request
だけ曲げる

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	arp op 1	sender hwaddr <VM hwaddr>	sender ipaddr <VM ipaddr>	target hwaddr 00:00:00:00:00:00	target ipaddr 169.254.169.254
-----------------------------	-------------	------------------------------	------------------------------	------------------------------------	----------------------------------

ARP Request

Metadata OVS 詳細 (ARP / 行き)

HV

```
table=20,  
arp,arp_tpa=169.254.169.254,arp_op=1,  
actions=  
move:vlan_vid[0..11]->NXM_NX_REG2[0..11],  
<Learn ARP>,  
resubmit(,25)
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

帰りのパケット用に
一時的なフローを作成

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	arp op 1	sender hwaddr <VM hwaddr>	sender ipaddr <VM ipaddr>	target hwaddr 00:00:00:00:00:00	target ipaddr 169.254.169.254
-----------------------------	-------------	------------------------------	------------------------------	------------------------------------	----------------------------------

ARP Request

Metadata OVS 詳細 (ARP / 行き)

HV

<Learn ARP> 詳細

```
learn(  
  table=15,idle_timeout=10,eth_type=0x0806,  
  arp_tpa=NXM_OF_ARP_SPA[],arp_op=2,  
  load:NXM_NX_REG2[0..11]->vlan_vid[0..11],  
  output:in_port),  
...
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

帰りのパケット用に
一時的なフローを作成

ovs
bond

Tenant
network

vlan id
<tenant vlan_id>

arp op
1

sender hwaddr
<VM hwaddr>

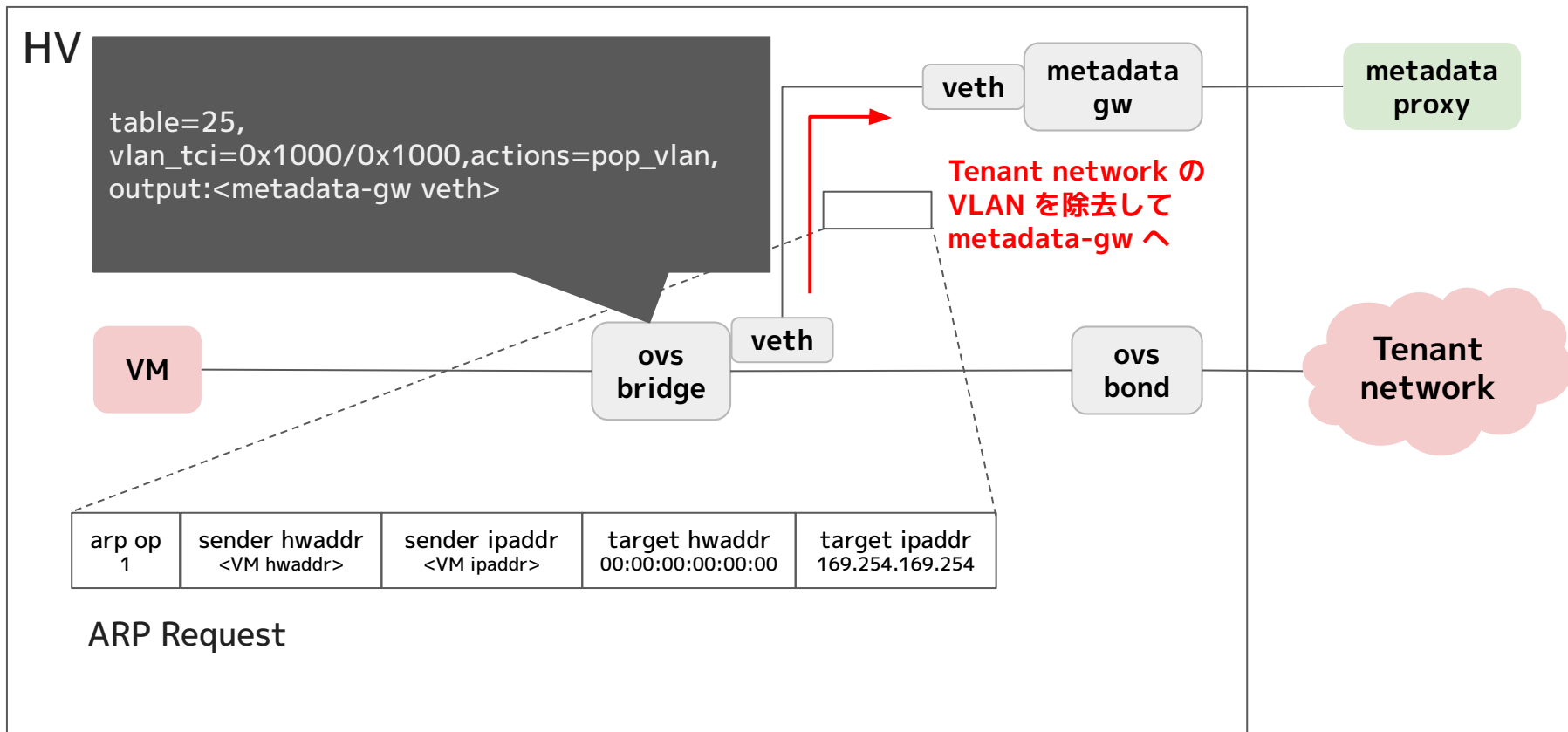
sender ipaddr
<VM ipaddr>

target hwaddr
00:00:00:00:00:00

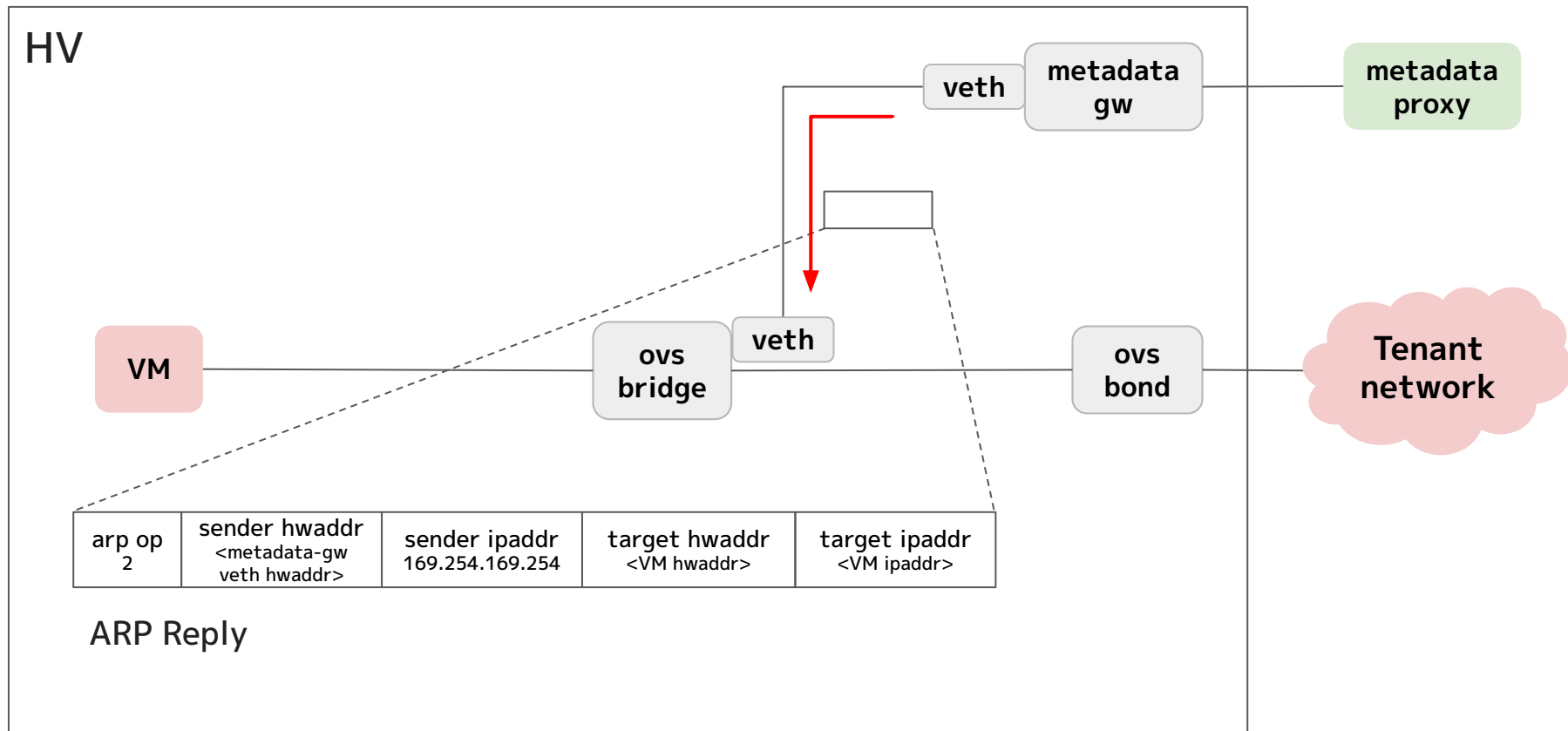
target ipaddr
169.254.169.254

ARP Request

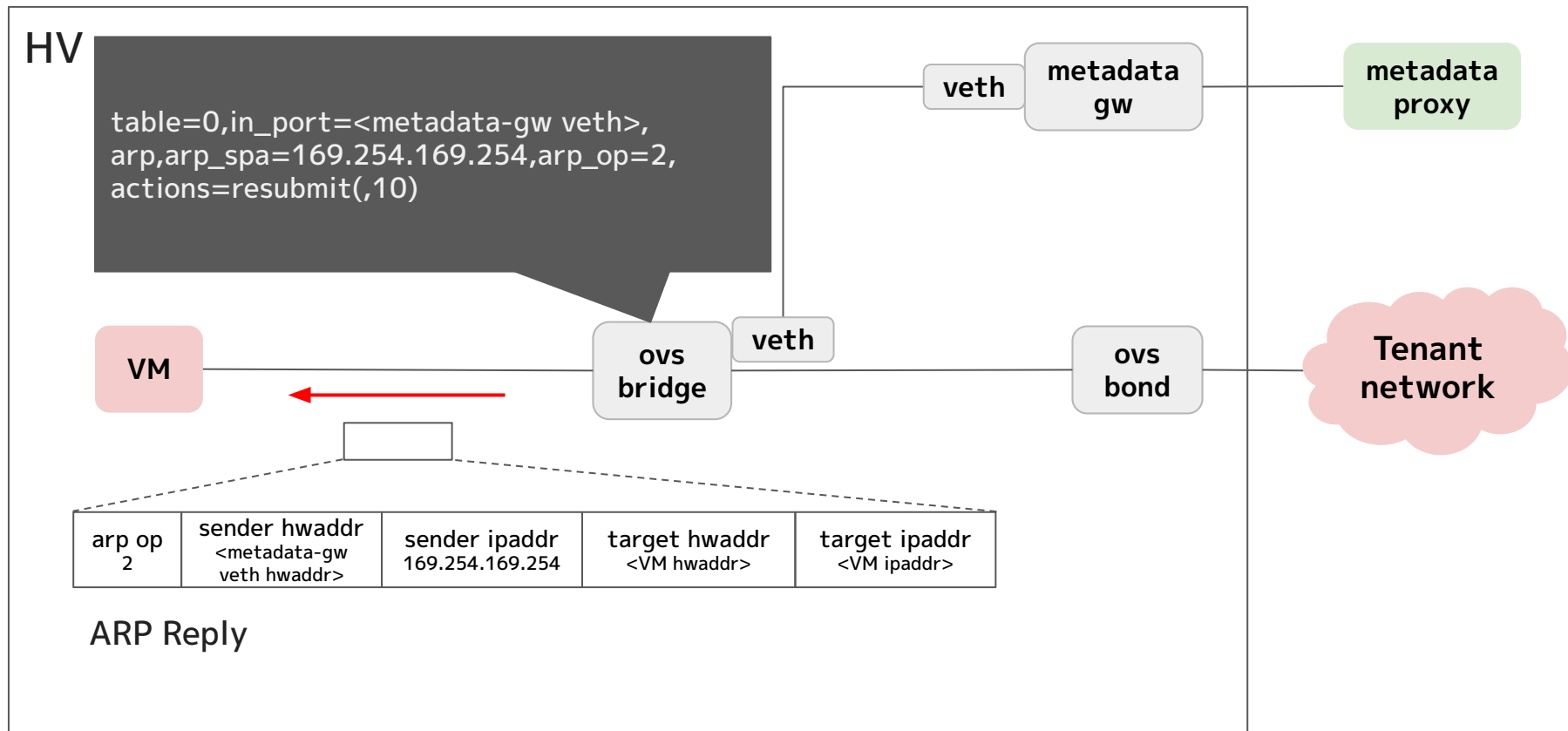
Metadata OVS 詳細 (ARP / 行き)



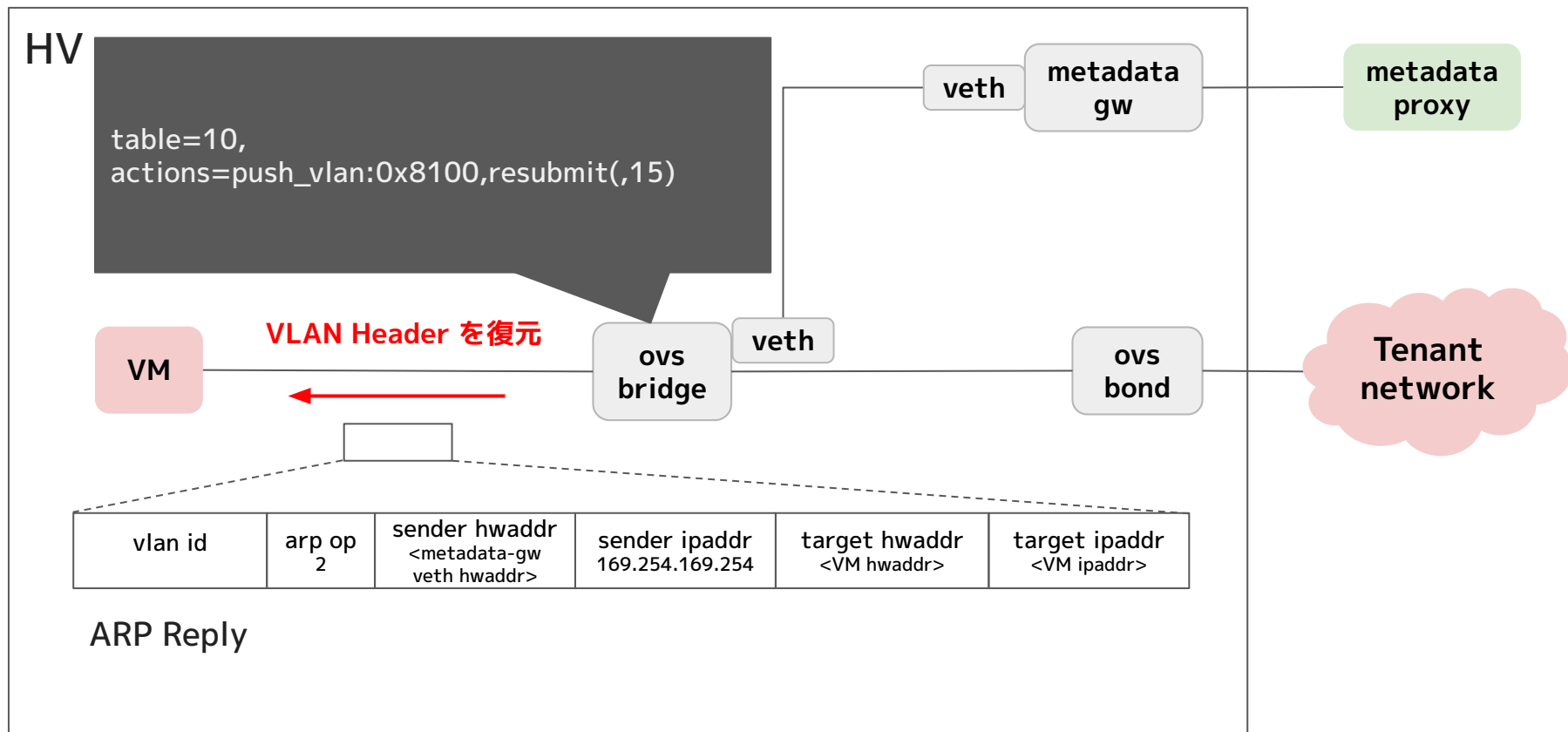
Metadata OVS 詳細 (ARP / 帰り)



Metadata OVS 詳細 (ARP / 帰り)



Metadata OVS 詳細 (ARP / 帰り)



Metadata OVS 詳細 (ARP / 帰り)

HV

```
table=15,idle_timeout=10,eth_type=0x0806,  
arp_tpa=<VM ipaddr>,arp_op=2,  
load:<tenant vlan_id>->vlan_vid[0..11],  
output:<VM port>
```

先ほど学習したフローで
VLAN ID を復元

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	arp op 2	sender hwaddr <metadata-gw veth hwaddr>	sender ipaddr 169.254.169.254	target hwaddr <VM hwaddr>	target ipaddr <VM ipaddr>
-----------------------------	-------------	---	----------------------------------	------------------------------	------------------------------

ARP Reply

Metadata OVS 詳細 (HTTP / 行き)

HV

```
$ ip route show
```

```
169.254.169.254 dev enp1s0  
proto dhcp scope link
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	dest ipaddr 169.254.169.254	src ipaddr <VM ipaddr>	dest port 80	src port *	HTTP Body
-----------------------------	--------------------------------	---------------------------	-----------------	---------------	--------------

HTTP Request

Metadata OVS 詳細 (HTTP / 行き)

HV

```
table=0,in_port=<VM port>,  
vlan_tci=0x1000/0x1000,  
tcp,nw_dst=169.254.169.254,tcp_dst=80,  
actions=resubmit(,20)
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

169.254.169.254
当てる HTTP Request
だけ曲げる

ovs
bond

Tenant
network

vlan id
<tenant vlan_id>

dest ipaddr
169.254.169.254

src ipaddr
<VM ipaddr>

dest port
80

src port
*

HTTP
Body

HTTP Request

Metadata OVS 詳細 (HTTP / 行き)

HV

```
table=20,  
tcp,nw_dst=169.254.169.254,tcp_dst=80,  
actions=  
move:vlan_vid[0..11]->NXM_NX_REG2[0..11],  
<Learn TCP>,  
resubmit(,25)
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

帰りのパケット用に
一時的なフローを作成

ovs
bond

Tenant
network

vlan id <tenant vlan_id>	dest ipaddr 169.254.169.254	src ipaddr <VM ipaddr>	dest port 80	src port *	HTTP Body
-----------------------------	--------------------------------	---------------------------	-----------------	---------------	--------------

HTTP Request

Metadata OVS 詳細 (HTTP / 行き)

HV

```
# <Learn TCP> 詳細
learn(
  table=15,idle_timeout=10,eth_type=0x0800,
  ip_dst=NXM_OF_IP_SRC[],ip_proto=6,
  tcp_dst=NXM_OF_TCP_SRC[],
  load:NXM_NX_REG2[0..11]->vlan_vid[0..11],
  output:in_port),
...
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

ovs
bond

Tenant
network

帰りのパケット用に
一時的なフローを作成

vlan id
<tenant vlan_id>

dest ipaddr
169.254.169.254

src ipaddr
<VM ipaddr>

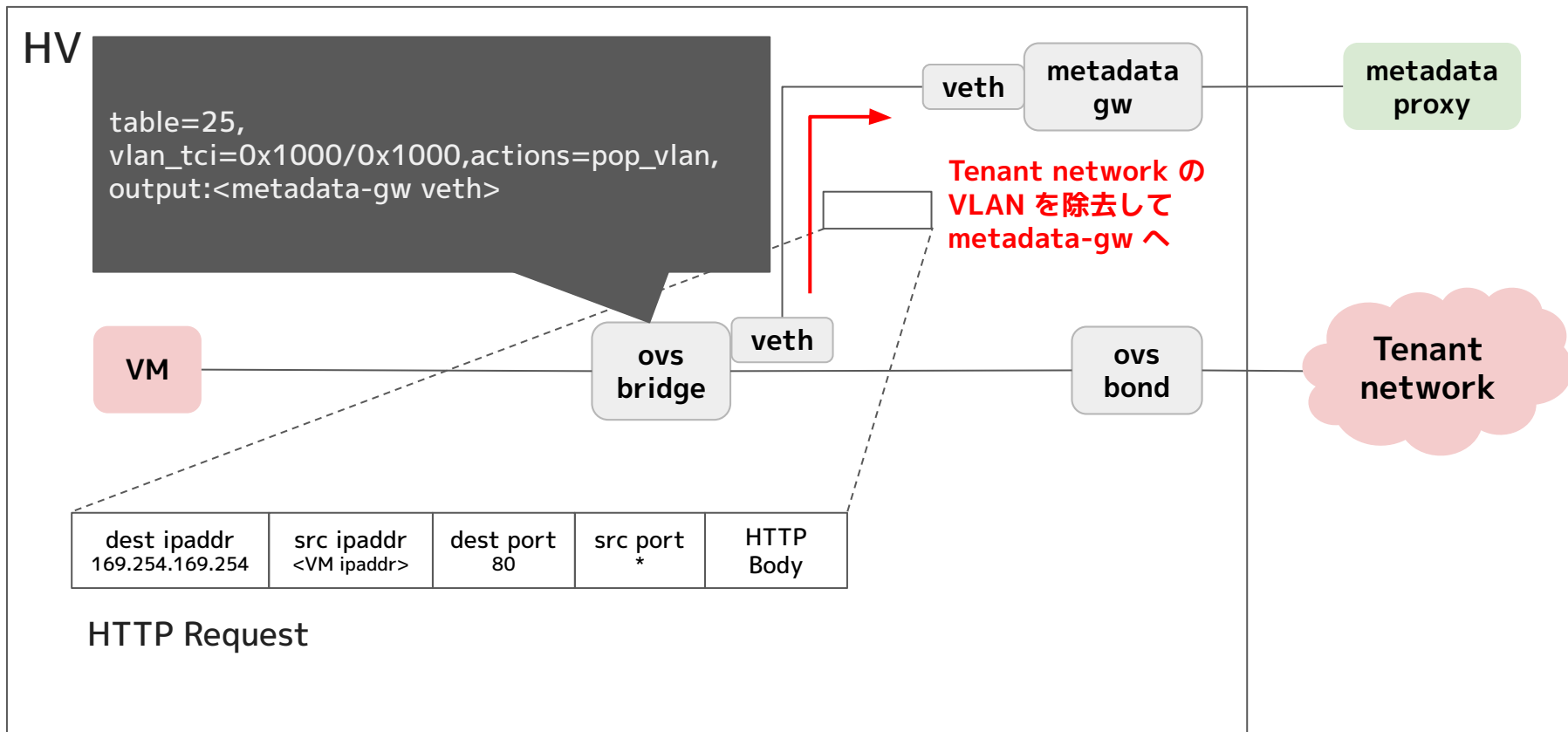
dest port
80

src port
*

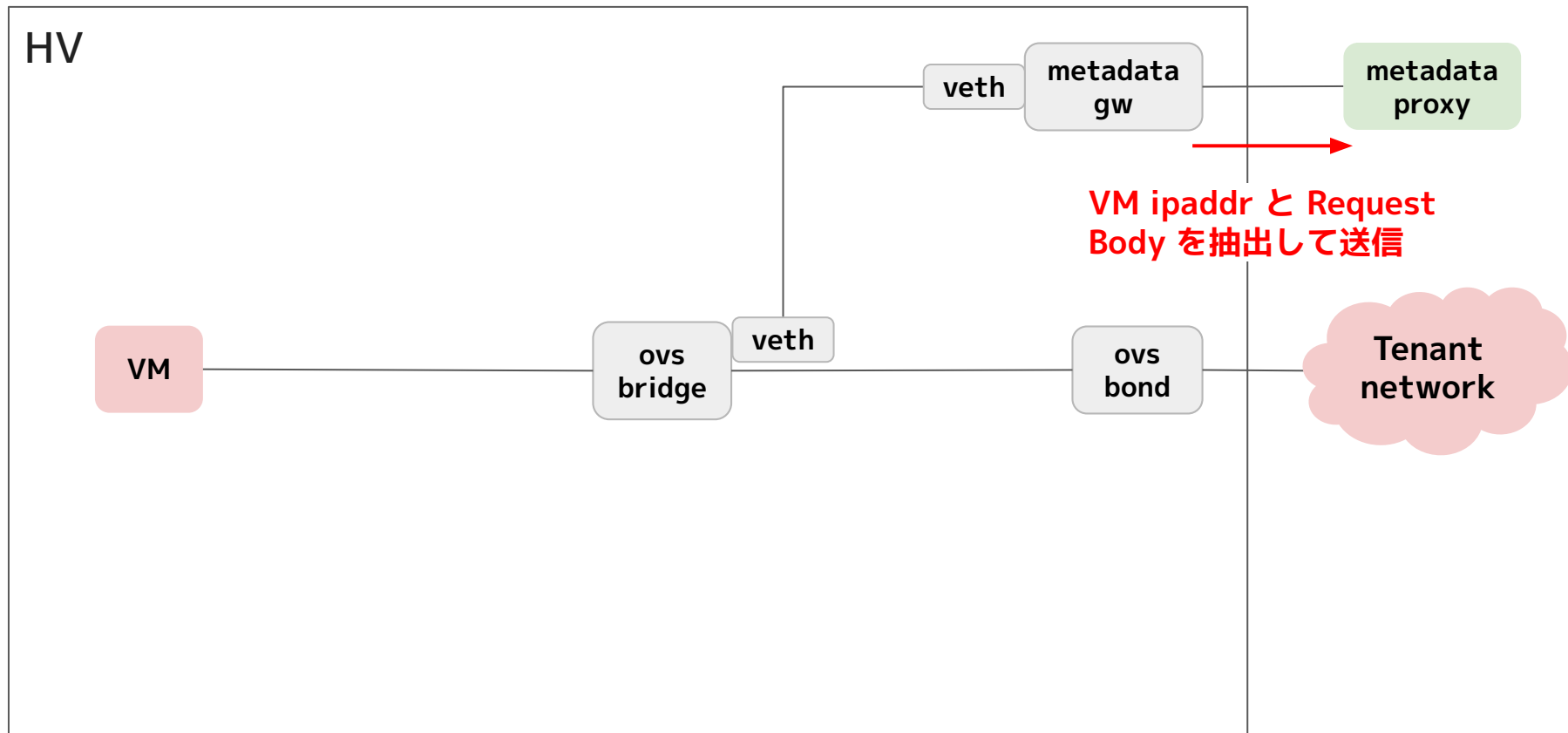
HTTP
Body

HTTP Request

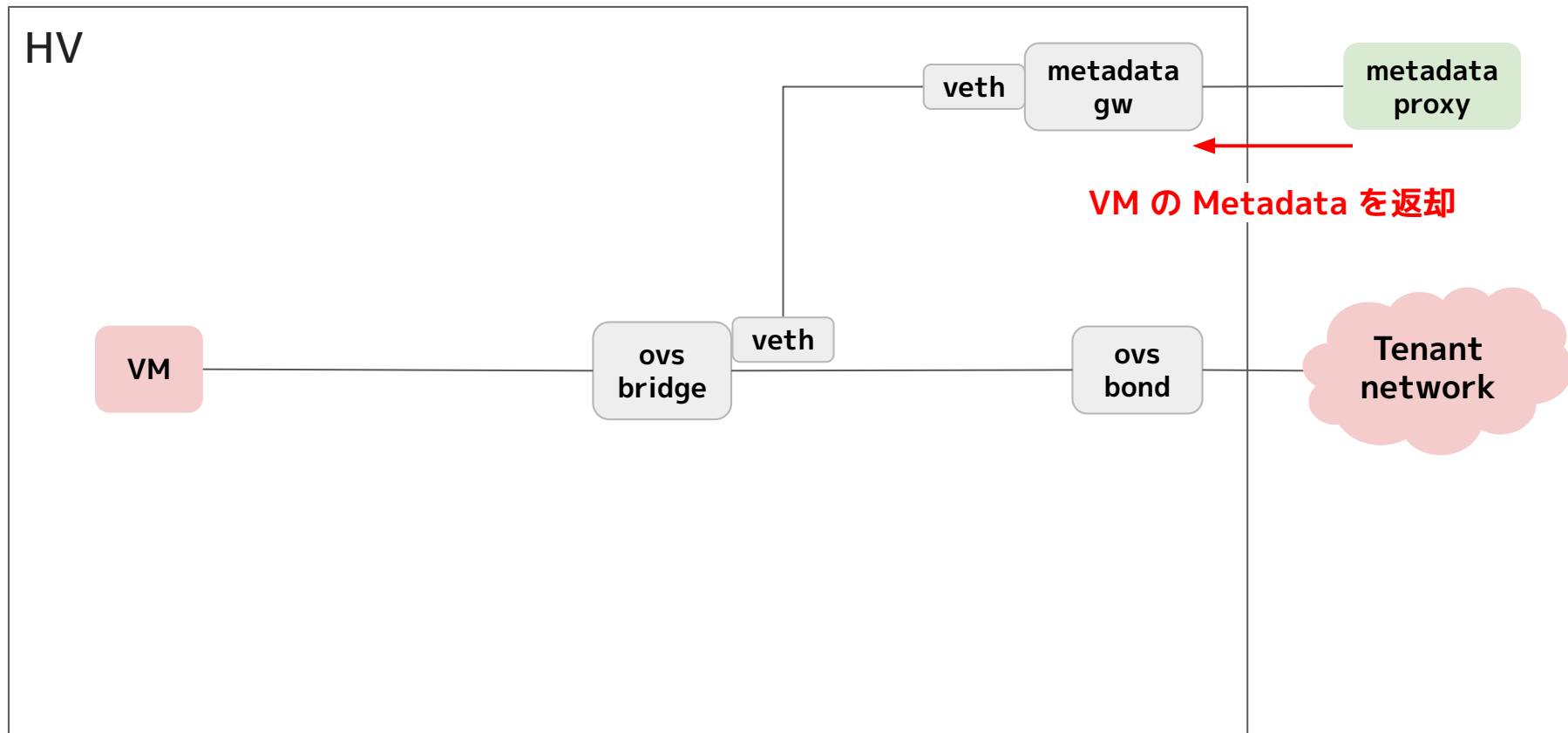
Metadata OVS 詳細 (HTTP / 行き)



Metadata OVS 詳細 (HTTP / 行き)



Metadata OVS 詳細 (HTTP / 帰り)



Metadata OVS 詳細 (HTTP / 帰り)

HV

```
$ ip rule show table metadata
30000: from 169.254.169.254 lookup
metadata proto static

$ ip route show table metadata
default dev veth-meta-br proto static scope
link
```

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

Linux の Source Routing により
metadata-gw からの帰りのパケット
を veth に曲げる

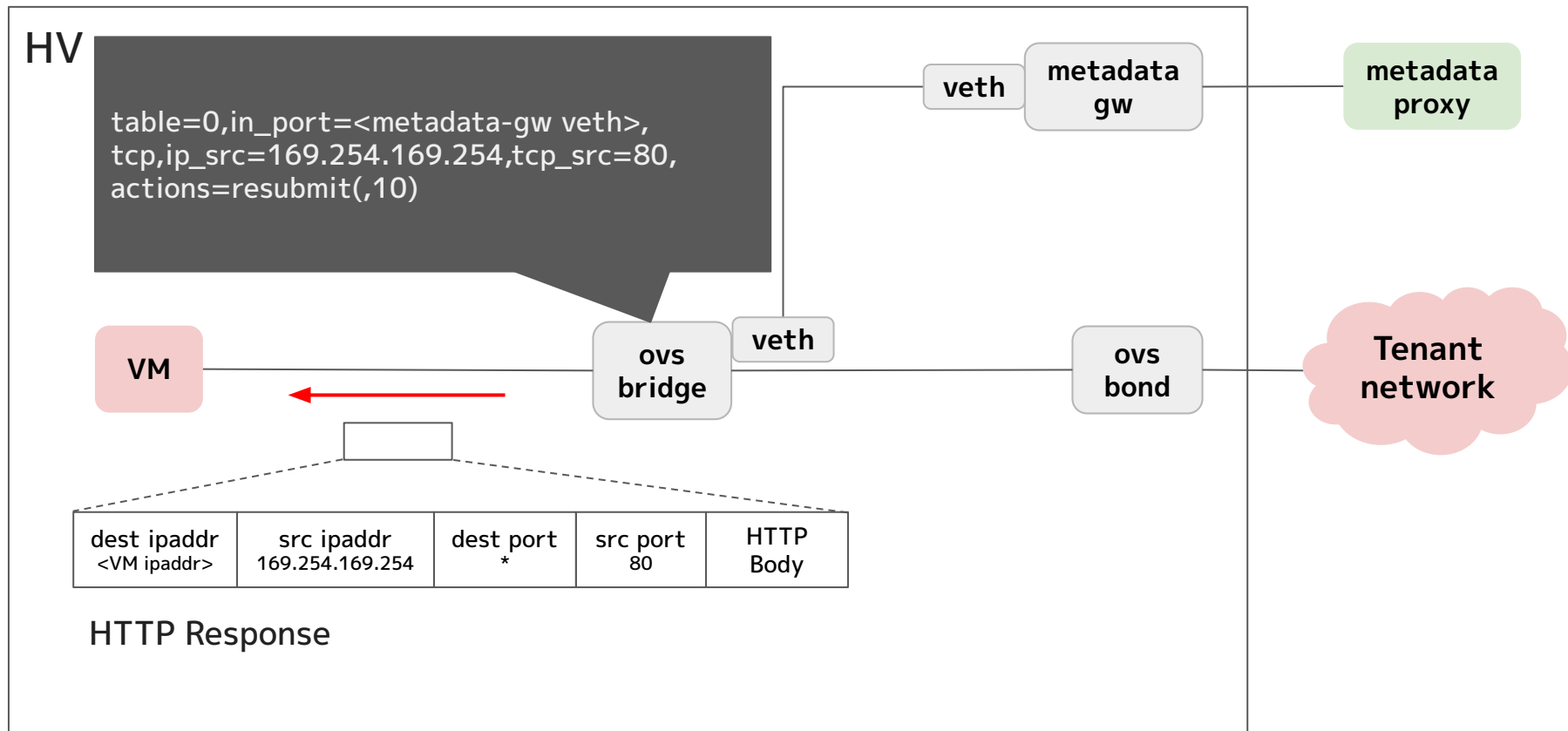
ovs
bond

Tenant
network

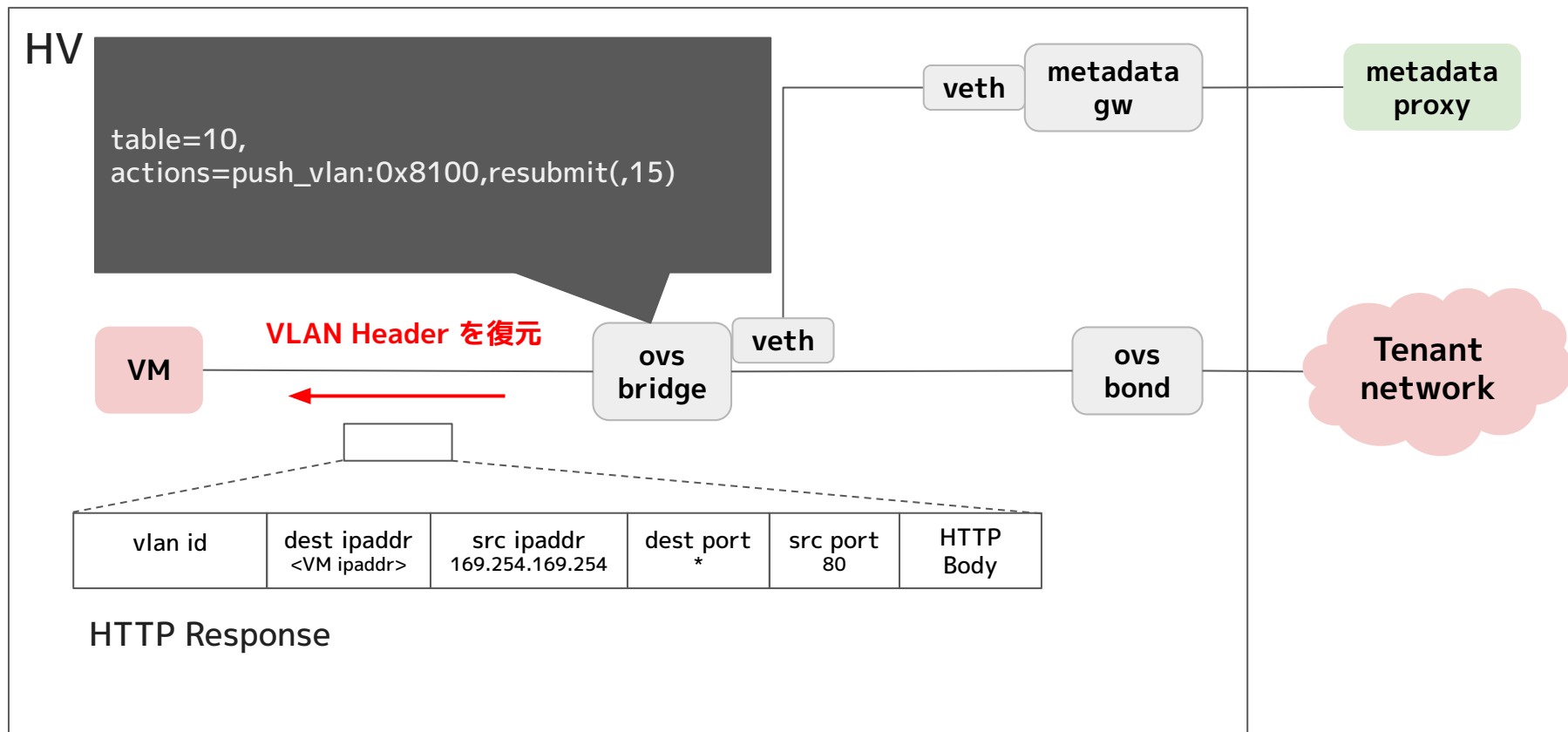
dest ipaddr <VM ipaddr>	src ipaddr 169.254.169.254	dest port *	src port 80	HTTP Body
----------------------------	-------------------------------	----------------	----------------	--------------

HTTP Response

Metadata OVS 詳細 (HTTP / 帰り)



Metadata OVS 詳細 (HTTP / 帰り)



sMetadata OVS 詳細 (HTTP / 帰り)

HV

```
table=15,idle_timeout=10,eth_type=0x0800,  
ip_dst=<VM ipaddr>,ip_proto=6,  
tcp_dst=<VM ephemeral port>,  
load:<tenant vlan_id>->vlan_vid[0..11],  
output:<VM port>
```

先ほど学習したフローで
VLAN ID を復元

VM

ovs
bridge

veth

veth

metadata
gw

metadata
proxy

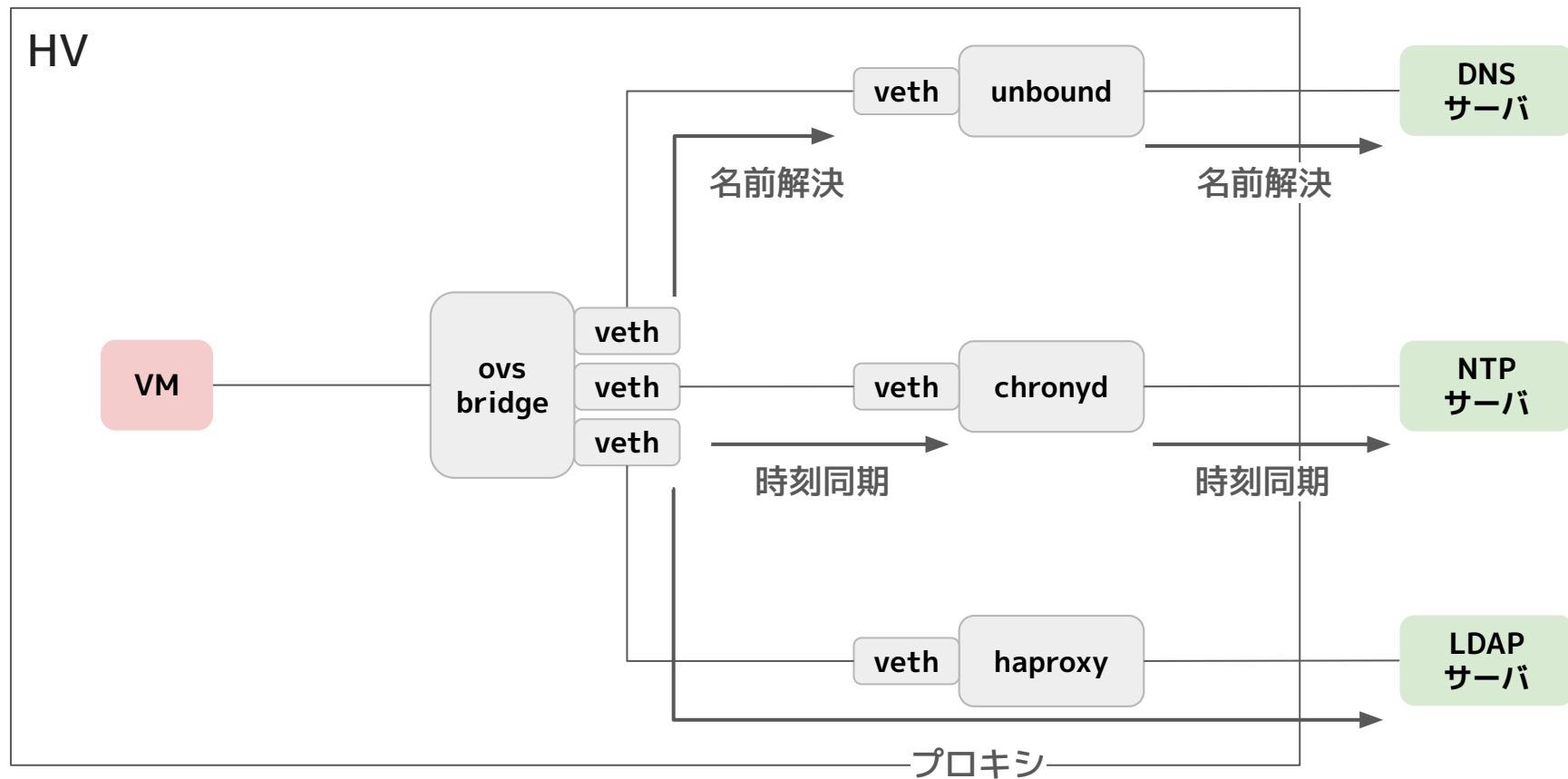
ovs
bond

Tenant
network

vlan id <tenant vlan_id>	dest ipaddr <VM ipaddr>	src ipaddr 169.254.169.254	dest port *	src port 80	HTTP Body
-----------------------------	----------------------------	-------------------------------	----------------	----------------	--------------

HTTP Response

DNS / NTP / LDAP への横展開



- HV 上で通信を折り返すことで、**テナント分離を崩さず VM からの undercloud 疎通を実現**
- DHCP / Metadata / DNS / NTP / LDAP 等の様々なサービスを HV 上のプロキシで適切に処理
- OVS Flow Rule を用いて**対象の通信だけを捕捉し**、VLAN 情報を保持・復元して返りの通信も可能に
- Linux Source Routing で**帰りの通信だけを** OVS データパスへ誘導

まとめ

- 高いテナント分離性を満たす必要がある制約の中で、OVS と Linux Source Routing を用いて undercloud との疎通性を確保する手法
 - OVS による柔軟なフォーワーディングやパケットの書き換え、データパスの設計
 - Source Routing によって特定の通信を OVS のデータパスに渡す手法

We are hiring!



一緒に働く仲間を大募集しています！

- プライベートクラウドや IaaS 基盤の開発・運用に興味がある方
- 仮想化基盤や仮想ネットワークの設計に携わることに意欲がある方

お気軽にお声がけください 🖐️

<https://it.cyberagent.group/team/ciu/>

<https://www.cyberagent.co.jp/way/list/detail/id=31427>

<https://www.cyberagent.co.jp/way/list/detail/id=31367>

<https://www.cyberagent.co.jp/way/list/detail/id=29497>