The Notorious M.T.U.

Kevin Benton - Mirantis
Sean Collins - Mirantis
Ihar Hrachyshka - Red Hat
Matt Kassawara - IBM
Objectives

- Learn about MTU in physical networks
- Learn about nuances of virtual networks that impact MTU
- Review confusing MTU options and workarounds/hacks in releases prior to Mitaka
- Apply MTU knowledge to reveal issues in OpenStack (neutron and nova) including several common deployment cases
- Learn about MTU solution in Mitaka
What is MTU?

- Largest network layer (3) data unit that underlying data link layer (2) can pass between transmitter and receiver
  - Commonly, the largest IP packet that can fit into available Ethernet frame
  - Layer 3 must dynamically adjust to changes at layer 2
- Typically 1500 bytes for 802.3 (Ethernet), although many devices support “jumbo frames” up to approximately 9000 bytes
- Provider/carrier network devices often support over 9000 bytes to account for overhead from MPLS, 802.1ad (Q-in-Q), etc.
IP Path MTU Discovery (PMTUD)

● Automatically determines the smallest MTU of network segments between transmitter and receiver
● Operates at IP layer using ICMP
  ○ Routers (3), not switches (2), handle MTU changes between segments
  ○ ICMP must pass freely between endpoints!
IP PMTUD - IPv4

- IPv4 supports fragmentation, but it can impact performance
- Operation
  - Transmitter generates a packet using the MTU of the underlying network interface and sets “Don’t Fragment” (DF) bit
  - If a segment between transmitter and receiver contains a smaller MTU, the router prior to that segment returns an ICMP “Fragmentation Needed” (Type 3, Code 4) message to the sender that contains the smaller MTU value
  - Operating system tracks MTU value for the receiver
  - Transmitter generates packet again using the smaller MTU and sets DF bit
  - Cycle repeats until the transmitter discovers the smallest MTU value between transmitter and receiver
IP PMTUD - IPv6

- IPv6 does not support fragmentation
- Operation
  - Transmitter generates a packet using the MTU of the underlying network interface
  - If a segment between transmitter and receiver contains a smaller MTU, the router prior to that segment returns an ICMP “Packet Too Big” (Type 2) message to the sender that contains the smaller MTU value
  - Operating system tracks MTU value for the receiver
  - Transmitter generates packet again using the smaller MTU
  - Cycle repeats until the transmitter discovers the smallest MTU value between transmitter and receiver
MTU changes at layer 2 = bad
MTU changes at layer 3 = good
Virtual networks and MTU

- **Flat**
  - Uses IEEE 802.3 (Ethernet)
  - Each flat network requires a unique physical network
  - Instance (VM) network interface can use underlying physical network MTU

- **VLAN**
  - Uses IEEE 802.1q (Ethernet with VLAN tagging)
    - Adds 32-bit field to Ethernet header containing a 12-bit VLAN ID and some other information
    - Effectively adds 4 bytes to Ethernet frame
    - Does not impact payload size
    - Multiple logical networks, each using a unique VLAN ID, can share a physical network
  - Instance (VM) network interface can use underlying physical network MTU
Virtual networks and MTU

- **Overlay**
  - Uses an encapsulation protocol such as VXLAN or GRE to pass arbitrary 802.3 Ethernet frames or IP packets via IP (and sometimes TCP/UDP)
  - Outer (native) IP headers, sometimes TCP/UDP headers, and protocol metadata create overhead that consumes a portion of the outer IP packet, thus reducing space available to devices using the overlay network
  - Instance (VM) network interface must use underlying physical network MTU minus the overhead
VXLAN protocol

- Uses UDP
- Encapsulates inner 802.3 Ethernet frame
- Calculate overhead for IPv4 using 1500-byte MTU
  - Subtract outer IP header (20 bytes) = 1480 bytes
  - Subtract UDP header (8 bytes) = 1472 bytes
  - Subtract VXLAN header (8 bytes) = 1464 bytes
  - Subtract inner 802.3 Ethernet header (14 bytes) = 1450 bytes for IP available to device using overlay network
GRE protocol

- Uses unique transport protocol (47)
- Encapsulates inner 802.3 Ethernet frame
- Calculate overhead for IPv4 using 1500-byte MTU
  - Subtract outer IP header (20 bytes) = 1480 bytes
  - Subtract GRE header (8 bytes) = 1472 bytes
  - Subtract inner Ethernet header (14 bytes) = 1458 bytes for IP available to device using overlay network

![Diagram](image)
Interesting observations

- **Linux**
  - Automatically configures tunnel network interface MTU by subtracting overlay protocol overhead from the underlying physical network interface MTU
  - Automatically configures bridge network interface MTU to use the lowest MTU of all ports (devices) on the bridge
  - Permits ends of virtual Ethernet (veth) pairs to use different MTUs

- **Open vSwitch**
  - Internally uses arbitrarily large MTU
  - Ignores MTU of bridge interface on host

Diagram:
- OVS (No MTU)
  - Silently discards packets larger than 1500 to this interface

- Linux Bridge (MTU 1500)
  - Forces downgrade of entire bridge

- Linux Bridge (MTU 9000)
  - 1500
  - 9000

- Open vSwitch (MTU 9000)
  - 1500
OpenStack MTU problems

● Neutron lacks obvious and consistent support for MTUs larger than 1500 bytes
● By default, nova creates security group bridges and interfaces using a 1500-byte MTU
● Features claiming to address MTU involve confusing and often useless options
  ○ advertise_mtu (neutron core)
  ○ physical_network_mtu (ML2 plug-in)
  ○ path_mtu (ML2 plug-in)
  ○ segment_mtu (ML2 plug-in)
  ○ veth_mtu (Open vSwitch agent)
  ○ network_device_mtu (neutron and nova core)
● Only some plug-ins support the MTU API extension
● Documentation… what documentation?
OpenStack MTU hacks

● Folsom to Juno
  ○ [Environment] Implement MTU larger than 1500 bytes on underlying physical network while leaving virtual network components at 1500 bytes to account for overlay protocol overhead
    ■ Instances on any network can use 1500 bytes
  ○ [Neutron] Manually configure Dnsmasq to provide a smaller MTU that accounts for overlay protocol overhead
    ■ Also reduces MTU for instances on flat and VLAN networks
  ○ [Neutron/Nova] Attempt to use the `network_device_mtu` option to configure MTU of virtual network components
    ■ Implementation varies by release, plug-in/agent, network types, and combination of other options
  ○ [Neutron] For the Open vSwitch plug-in/agent with veth interfaces, attempt to use the `veth_mtu` option
OpenStack MTU hacks

● Kilo and Liberty
  ○ [Neutron+ML2] Configure Dnsmasq to provide a smaller MTU that accounts for overlay protocol overhead
    ■ Combination of $\text{path\_mtu}$ and $\text{advertise\_mtu}$ options
    ■ Only impacts instances on overlay networks
  ○ [ML2] Attempt to use variety of additional options that configure MTU for some but not all virtual network components
    ■ $\text{segment\_mtu}$
    ■ $\text{physical\_network\_mtus}$
  ○ [Neutron/Nova] Attempt to use the $\text{network\_device\_mtu}$ option with or without additional options
Common use cases

- Assume proper configuration of underlying physical network
- Assume use of Liberty
- Assume VXLAN overlay networks with IPv4 endpoints
  - 50 bytes of overhead
- Cases 1-4 only use `path_mtu` and `advertise_mtu` options, if available, to configure instance network interface MTU
- Cases 5-6 also use the `network_device_mtu` option
Case 1: Open vSwitch agent with 1500-byte MTU

advertise_mtu = true and path_mtu = 1500
Case 2: Open vSwitch agent with 9000-byte MTU

advertise_mtu = true and path_mtu = 9000
Case 3: Linux bridge agent with 1500-byte MTU

advertise_mtu = true and path_mtu = 1500
Case 4: Linux bridge agent with 9000-byte MTU

advertise_mtu = true and path_mtu = 9000
Case 5: Open vSwitch agent with 9000-byte MTU

`network_device_mtu = 9000`
Case 6: Linux bridge agent with 9000-byte MTU

network_device_mtu = 9000
OpenStack MTU solution (Mitaka+)

● **Neutron**
  ○ Replace variety of options with a single option suitable for most environments
  ○ Consistently calculate and set appropriate MTU for all virtual network components
  ○ By default, provide useful (non-zero) MTU value in API

● **Nova**
  ○ Use the MTU value that neutron provides via RPC for security group bridges and interfaces

● **os-vif library**
  ○ Replaces nova VIF code
  ○ Contains essentially the same MTU implementation that currently exists in nova
OpenStack MTU solution (Mitaka+)

- Implementation details
  - Move `segment_mtu` option from ML2 to neutron and rename to `global_physnet_mtu`
    - Resides in `[DEFAULT]` section
    - Visible to all plug-ins
    - Change default value from 0 to 1500
    - Yields calculation of correct MTU for virtual network components in nearly all environments
  - By default, enable `advertise_mtu` option in neutron
    - Provides correct MTU to instances via DHCP (IPv4) or RA (IPv6)
  - Deprecate `path_mtu` option in ML2
    - Neutron review #302089
  - Keep `path_mtu` and `physical_network_mtus` options in ML2
    - Supports rare environments that implement unique MTU value for each underlying physical or logical network
OpenStack MTU solution (Mitaka+)

- Not all rainbows and unicorns
  - The `global_physnet_mtu` option came after a separate effort to use “sane” values for other MTU options. As a result, the `path_mtu` value currently overrides the `global_physnet_mtu` value for overlay networks.
    - Use the same value for `global_physnet_mtu` and `path_mtu`
    - See neutron [review #308989](https://www.openstack.org/review/#308989)
  - Does not recalculate MTU for existing virtual networks
    - Manually update MTU values in the database
    - Only impacts new devices belonging to the same virtual network
    - Use with caution

- For your sanity, use single consistent MTU value for entire underlying physical network
But I can’t switch to Mitaka!

- Backporting primary resolution to Liberty
  - Nova [review #285710](#)
  - Neutron [review #305782, review #308229](#)
  - Requires using ML2 and the variety of additional options introduced in Kilo

- In addition to Liberty backports
  - [Neutron] Enable `advertise_mtu`
  - [ML2] set `segment_mtu` to reference underlying physical network MTU
    - Note location and name change for upgrade purposes
  - [Neutron/Nova] Unset `network_device_mtu`
  - [Neutron] Update ‘mtu’ column in ‘networks’ table and recreate networks
What about Kilo and earlier releases?
What about Kilo and earlier releases?

Seriously, plan an upgrade. OpenStack, especially Neutron, has come a long way in just a few releases.
Next steps

● Recalculate MTU for existing networks
  ○ Bug #1556182

● Remove `network_device_mtu` option from neutron and nova
  ○ Currently deprecated in nova

● Adopt os-vif to communicate MTU values between neutron and nova

● Deployment tools should remove MTU hacks
Questions?