

Network Bandwidth Isolation

linux.conf.au Brisbane 2011

Simon Horman <simon@horms.net>
Horms Solutions Ltd., Tokyo

Australia Day, 26th January 2011

Outline

- Part I: Overview
- Part II: Identifying Packets
- Part III: Packet Scheduling
- Part IV: Interesting Problems

Part I

Overview

Fairness

- Wish to ensure that each domain received a fair share of network-related resources
 - As defined by the administrator
- Guard against malicious domains
- Guard against domains that have been infected by a virus

Assumptions

Frame for discussion

- Xen — Though this ought to be applicable to KVM
- Network is bridged in dom0
- Dom0 is running Linux
- Only discuss transmit path

Network-Related Resources

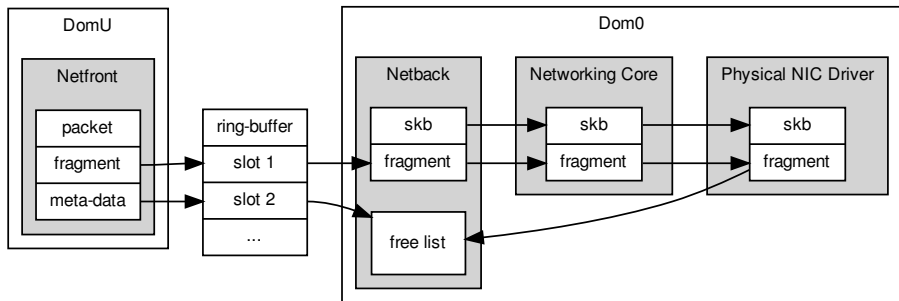
- NIC Bandwidth
 - How fast packets are being transmitted and received by domUs
- Dom0 CPU
 - How fast packets are being transmitted and received by domUs
- Dom0 Kernel memory
 - How many packets are held in the kernel

Packet Scheduling

- Prioritise packets based on domain
 - NIC Bandwidth
 - Dom0 CPU
- Drop packets if a domain has too many enqueued
 - Dom0 Kernel memory usage

Netback/Netfront Flow Control

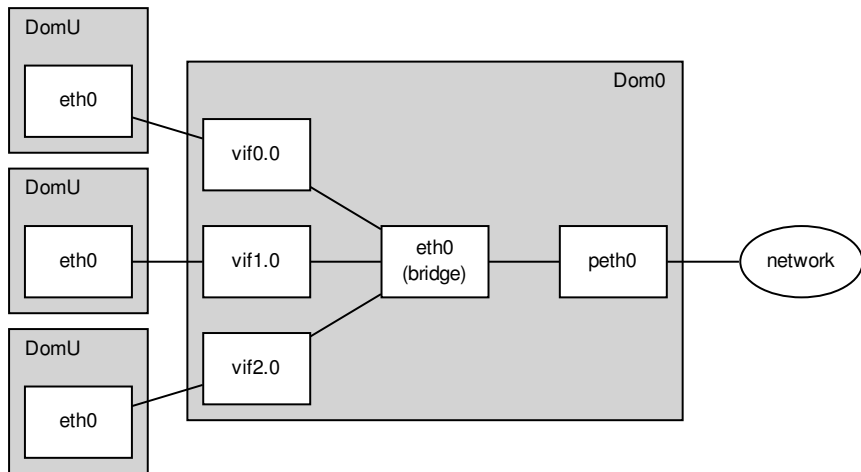
- End-to-end flow control from netfront until a packet is transmitted by the destination interface
- Allows packet scheduling to control network-related resource usage
 - dom0 CPU
 - dom0 Kernel memory



Part II

Identifying Packets

Bridged Xen Network



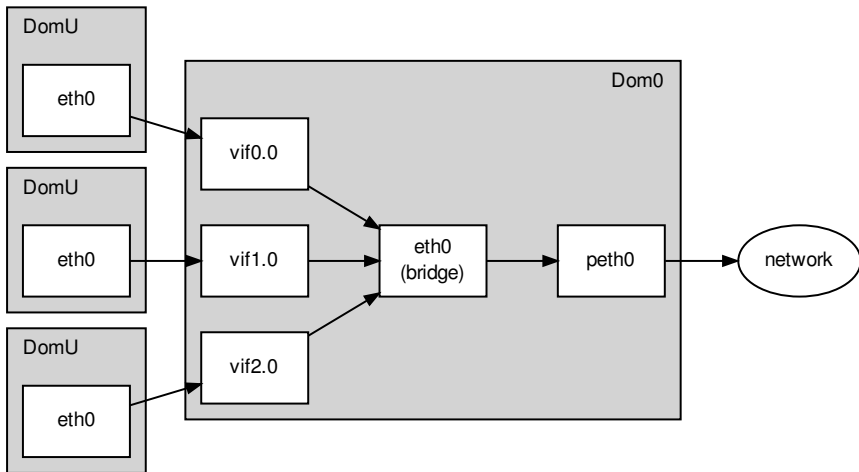
Three domUs bridged to a single physical interface

Tools of the trade

- `iptables`
 - Can mark packets passing through interfaces
 - Keys can include source MAC address and interface
- `net_cls`
 - Can mark packets based on their task



DomU Transmit: Identifying Packets



- Match the interface from which packets enter eth0 (bridge)
- Identifies the source-domU

DomU Transmit: iptables Rules

Mark the packets according to which interface they arrive on

```
iptables -t mangle -A FORWARD -m physdev \  
    --physdev-in vif2.0 -j MARK --set-mark 100  
iptables -t mangle -A FORWARD -m physdev \  
    --physdev-in vif3.0 -j MARK --set-mark 110  
iptables -t mangle -A FORWARD -m physdev \  
    --physdev-in vif5.0 -j MARK --set-mark 120
```

Part III

Packet Scheduling

Packet Scheduling

- Filter
 - Assign to a class
- Prioritise
 - Based on class assignment
 - May selectively delay packets
- Queue
 - For transmission after filtering or prioritisation
- Drop
 - If a queue becomes full

How many packets are held in the dom0 kernel

- Limited by the number of netback ring-buffer slots

$$p \leq n$$

where: p : transmit packets enqueued in dom0 for vif $N.M$
 n : netback ring-buffer slots (default = 256)

How fast packets are transmitted

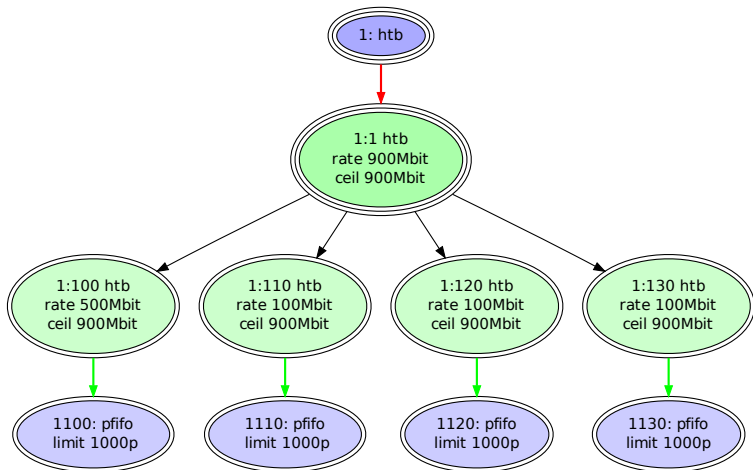
- Delaying packets in dom0 should be sufficient
- Dropping packets may actually be harmful
 - Holding onto packets actually slows down domU

Allow classes exceed their rate if there is unused bandwidth

- *rate*: Maximum rate a class and its children are guaranteed
- *ceil*: Maximum rate at which a class can send, if its parent has bandwidth to spare

tc-htb(8) man page

DomU Transmit: Packet Scheduling Hierarchy



Tools of the Trade

- tc
 - Used to configure traffic control
 - Configure filters
 - Configure packet scheduling



Root Class

```
tc qdisc add dev peth0 root handle 1: htb default 130
```

Inner Class

- To allow Borrowing

```
tc class add dev peth0 parent 1: classid 1:1 htb \  
rate 900Mbit ceil 900Mbit
```

Leaf Classes

- One per domain + default

```
tc class add dev peth0 parent 1:1 classid 1:100 htb \  
    rate 500Mbit ceil 900Mbit  
tc class add dev peth0 parent 1:1 classid 1:110 htb \  
    rate 100Mbit ceil 900Mbit  
tc class add dev peth0 parent 1:1 classid 1:120 htb \  
    rate 100Mbit ceil 900Mbit  
tc class add dev peth0 parent 1:1 classid 1:130 htb \  
    rate 100Mbit ceil 900Mbit
```

DomU Transmit: FIFO Rules

Terminate each leaf class with a fifo

- The default is a PFIFO, made explicit by the following rules

```
tc qdisc add dev peth0 parent 1:100 handle 1100: \  
    pfifo limit 1000  
tc qdisc add dev peth0 parent 1:110 handle 1110: \  
    pfifo limit 1000  
tc qdisc add dev peth0 parent 1:120 handle 1120: \  
    pfifo limit 1000  
tc qdisc add dev peth0 parent 1:130 handle 1130: \  
    pfifo limit 1000
```

Filter based on the fwmark set by iptables

- handle N is the fwmark match
- flowid X:Y is the class to assign the packet to match

```
tc filter add dev peth0 protocol ip parent 1: \  
    handle 100 fw flowid 1:100  
tc filter add dev peth0 protocol ip parent 1: \  
    handle 110 fw flowid 1:110  
tc filter add dev peth0 protocol ip parent 1: \  
    handle 120 fw flowid 1:120
```


Part IV

Interesting Problems

Problem 1: UDP, VLANs and Lack of Flow Control

Problem

- VLAN devices do not support scatter-gather
- This means that each skb needs to be linearised and thus cloned if they are transmitted on a VLAN device
- Cloning results in the original fragments being released
- This breaks Xen's netfront/netback flow-control

Result

- A guest can flood dom0 with packets
- Very effective DoS attack on dom0 and other domUs

Problem 1: UDP, VLANs and Lack of Flow Control

Work-Around

- Use the credit scheduler to limit the rate of a domU's virtual interface to something close to the rate of the physical interface

```
vif = [ "mac=00:16:36:6c:81:ae,bridge=eth4.100,  
        script=vif-bridge,rate=950Mb/s" ]
```

- Still uses quite a lot of dom0 CPU if domU sends a lot of packets
- But the DoS is mitigated

Problem 1: UDP, VLANs and Lack of Flow Control

Partial Solution

- scatter-gather enabled VLAN interfaces
- Problem is resolved for VLANs with supported physical devices
- Still a problem for any other device that doesn't support scatter-gather

Problem 1: UDP, VLANs and Lack of Flow Control

Patches

- Included in v2.6.26-rc4
 - "Propagate selected feature bits to VLAN devices" and;
 - "Use bitmask of feature flags instead of separate feature bit" by Patrick McHardy.
 - "igb: allow vlan devices to use TSO and TCP CSUM offload" by Jeff Kirsher
- Patches for other drivers have also been merged

<http://kerneltrap.org/mailarchive/linux-netdev/2008/5/21/1898674>

<http://kerneltrap.org/mailarchive/linux-netdev/2008/5/23/1922094>

<http://kerneltrap.org/mailarchive/linux-netdev/2008/6/5/2037984>

Problem 2: Bonding and Lack of Queues

Problem

- The default queue on bond devices is no queue
 - This is because it is a software device, and generally queuing doesn't make sense on software devices
- qdiscs default the queue length of the device they are attached to

Result

- It was observed that netperf TCP_STREAM only achieves 45-50Mbit/s when controlled by a class with a ceiling of 450Mbit/s
- A 10x degradation!

Problem 2: Bonding and Lack of Queues

Solution 1a

- Set the queue length of the bonding device before adding qdiscs

```
ip link set txqueuelen 1000 dev bond0
```

Solution 1b

- Set the queue length of the qdisc explicitly

```
tc qdisc add dev bond0 parent 1:100 handle 1100: \  
    pfifo limit 1000
```

Problem 3: TSO and Lack of Accounting Accuracy

Problem

- If a packet is larger than the MTU of the class it is accounted as being approximately the size of the MTU
- And the giants counter for the class is incremented
- In this case, the default MTU is 2047 bytes
- But TCP Segmentation Offload (TSO) packets can be much larger
 - 64kbytes
 - By default Xen domUs will use TSO

Result

- The result similar to no bandwidth control of UDP

Problem 3: TSO and Lack of Accounting Accuracy

Details

- `ceil_log` is a logarithmic scaling value used when accounting the cost of a packet.

```
mtu = 2047;                                #default
cell_log = 0;
while (mtu >> cell_log) > 255)
    cell_log++;
```

- In this example, `cell_log` is 3

Code has been simplified for the sake of brevity

Problem 3: TSO and Lack of Accounting Accuracy

Details

- rtab is a lookup table of packet costs

```
for (i = 0; i < 256; i++) {  
    size = (i + 1) << cell_log;  
    rtab[i] = TIME_UNITS_PER_SEC * size / rate;  
}
```

- rtab is looked up using `packet_size >> cell_log` as the index
- Where the index is truncated to 255

Code has been simplified for the sake of brevity

Problem 3: TSO and Lack of Accounting Accuracy

Details

packet size	ratab index	rtab value
1–15	1	16n
16–23	2	24n
24–32	3	32n
⋮	⋮	⋮
2032–2039	254	2032n
2040–	255	2040

Using the abridged algorithm on the previous slides

Using $\text{cell_log} = 3$

Problem 3: TSO and Lack of Accounting Accuracy

Workaround 1

- Disable TSO in the guest

```
# ethtool -k eth0 | grep "tcp segmentation offload"  
tcp segmentation offload: on  
# ethtool -K eth0 tso off  
# ethtool -k eth0 | grep "tcp segmentation offload"  
tcp segmentation offload: off
```

- ...but the guest can re-enable it

Problem 3: TSO and Lack of Accounting Accuracy

Workaround 2

- Set the MTU of classes to 40000

```
tc class add dev peth2 parent 1:1 classid 1:101 \  
    rate 10Mbit ceil 950Mbit mtu 40000
```

- Large enough to give sufficient accuracy
- Larger values will result in a loss of accuracy when accounting smaller packets

<http://kerneltrap.org/mailarchive/linux-netdev/2009/11/2/6259456>

Problem 3: TSO and Lack of Accounting Accuracy

Solution

- Account for large packets
- Instead of truncating the index, use rtab values multiple times
$$\text{rtab}[255] * (\text{index} \gg 8) + \text{rtab}[\text{index} \& 0xFF]$$
- "Make HTB scheduler work with TSO" by Ranjit Manomohan was included in 2.6.23-rc1

<http://kerneltrap.org/mailarchive/linux-netdev/2007/12/11/488315>

Conclusion

- Existing infrastructure can be used for network bandwidth control
- The key is to be able to identify packets
- And then design an appropriate class hierarchy
- But there are some subtle traps — testing is vital

Questions