Control and forwarding plane separation
on an open-source router

Linux Kongress
2010-09-23 in Nürnberg

Robert Olsson, Uppsala University
Olof Hagsand, KTH
More than 10 year in production at Uppsala University

Stockholm

ISP/SUNET
AS1653

UU- 1

UU- 2

Full Internet routing via EBGP/IBGP

2 * XEON 5630
TYAN 7025
4 *10g ixgbe sfp+ LR/SR

L- green

AS 2834

L- red

IPv4/IPv6

Interneral UU-Net

OSPF

Local BGP peering in Uppsala

Now at 10g towards ISP, SFP+ 850 nm, 1310nm

DMZ
Motivation

- Separate control-plane from forwarding plane
  A la IETF FORCES
- Control-plane: sshd, bgp, stats, etc on CPU core 0
- Forwarding-plane: Bulk forwarding on core1,...,coreN
- This leads to robustness of service against overload and DOS attacks, etc
- Enabled by:
  multi-core CPUs
  NIC hw classifiers
  Fast Buses (QPI/PCI-E gen2)
Control-plane separation on a multi-core

Incoming traffic

Classifier

Control traffic

Forwarding traffic

Control traffic

CE (core0)

FE1 (core1)

FE2 (core2)

... 

FEN (coreN)

Control-Plane separation

Forwarding-Elements

Outgoing traffic

Router
Hi-End Hardware

XEON 2 x E5630
TYAN S7025 Motherboard
Intel 82599
Block hardware structure

CPU0
Quad-core

CPU1
Quad-core

IOH
Tylersburg

More I/O devices
Hi-End Hardware/Latency

![Graph showing latency results with stride 128](image)
Hardware - NIC

Intel 10g board Chipset 82599 with SFP+

Open chip specs. Thanks Intel!
The classification in the Intel 82599 consists of several steps, each is programmable. This includes:

- **RSS** (Receiver-side scaling): hashing of headers and load-balancing
- **N-tuples**: explicit packet header matches
- **Flow-director**: implicit matching of individual flows.
Routing daemons

Packet forwarding is done in Linux kernel Routing protocols is run in user-space daemons

Currently tested versions of quagga Bgp, OSPF both IPv4, IPv6 Cisco API
Experiment 1: flow separation external source

- Bulk forwarding data from source to sink (10Gb/s mixed packet lengths): mixed flow and packet lengths
- Netperf's TCP transactions emulated control data from a separate host
- Study latency of TCP transactions
N-tuple or Flowdirector

ethtool -K eth0 ntuple on

ethtool -U eth0 flow-type tcp4 src-ip 0x0a0a0a01 src-ip-mask 0xFFFFFFFF dst-ip 0 dst-ip-mask 0 src-port 0 src-port-mask 0 dst-port 0 dst-port-mask 0 vlan 0 vlan-mask 0 user-def 0 user-def-mask 0 action 0

ethtool -u eth0

N-tuple is supported by SUN Niu and Intel ixgbe driver.

Actions are: 1) queue 2) drop

But we were lazy and patched ixgbe for ssh and BGP to use CPU0
Even more lazy... we found the flow-director was implicitly programmed by outgoing flows. So both incoming and outgoing would use the same queue.

So if we set affinity for BGP, sshd etc we could avoid the N-tuple filters

Example:
taskset -c 0 /usr/bin/sshd

Neat....
RSS is still using CPU0

So we both got our “selected traffic”
Plus the bulk traffic from RSS

We just want RSS to use “other” CPU's
Patching RSS

Just a one-liner...

diff --git a/drivers/net/ixgbe/ixgbe_main.c b/drivers/net/ixgbe/ixgbe_main.c
index 1b1419c..08bbd85 100644
--- a/drivers/net/ixgbe/ixgbe_main.c
+++ b/drivers/net/ixgbe/ixgbe_main.c
@@ -2379,10 +2379,10 @@ static void ixgbe_configure_rx(struct ixgbe_adapter *adapter)
     mrqc = ixgbe_setup_mrqc(adapter);

     if (adapter->flags & IXGBE_FLAG_RSS_ENABLED) {
-       /* Fill out redirection table */
-       for (i = 0, j = 0; i < 128; i++, j++) {
+       /* Fill out redirection table but skip index 0 */
+       for (i = 0, j = 1; i < 128; i++, j++) {
           if (j == adapter->ring_feature[RING_F_RSS].indices)
             j = 0;
           j = 1;
           /* reta = 4-byte sliding window of
            * 0x00..(indices-1)(indices-1)00..etc. */
           reta = (reta << 8) | (j * 0x11);
## Patching RSS

<table>
<thead>
<tr>
<th>CPU-core</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of packets</td>
<td>0</td>
<td>196830</td>
<td>200860</td>
<td>186922</td>
<td>191866</td>
<td>186876</td>
<td>190106</td>
<td>190412</td>
</tr>
</tbody>
</table>

No traffic to CPU core 0 still RSS gives fairness between other cores.
Transaction Performance
netperf TCP RR

On “router”
taskset -c 0 netserver
Don't let forwarded packets program the flowdirector

A new one-liner patch....

@@ -5555,6 +5555,11 @@ static void ixgbe_atr(struct ixgbe_adapter *adapter, struct sk_buff *skb,
 u32 src_ipv4_addr, dst_ipv4_addr;
 u8 l4type = 0;
 +
 + if(!skb->sk) {
 +     /* ignore nonlocal traffic */
 +     return;
 + }
 +
 + /* check if we're UDP or TCP */
 if (iph->protocol == IPPROTO_TCP) {
     th = tcp_hdr(skb);
Instrumenting the flow-director

ethtool -S eth0 | grep fdir
Flow-director stats/1

```
fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 8191
fdir_coll: 0
fdir_match: 195
fdir_miss: 573632813   <--- Bulk forwarded data from RSS
fdir_ustat_add: 1      <--- Old ssh session
fdir_ustat_remove: 0
fdir_fstat_add: 6
fdir_fstat_remove: 0
fdir_maxlen: 0

ustat → user stats
fstat → failed stats
```
Flow-director stats/2

fdir_maxhash: 0
fdir_free: 8190
fdir_coll: 0
fdir_match: 196
fdir_miss: 630653401
fdir_ustat_add: 2        <-- New ssh session
fdir_ustat_remove: 0
fdir_fstat_add: 6
fdir_fstat_remove: 0
Flow-director stats/3

fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 8190
fdir_coll: 0
fdir_match: 206 <-- ssh packets are matched
fdir_miss: 645067311
fdir_ustat_add: 2
fdir_ustat_remove: 0
fdir_fstat_add: 6
fdir_fstat_remove: 0
Flow-director stats/4

fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 32768  <-- Now increased 32k
fdir_coll: 0
fdir_match: 0
fdir_miss: 196502463
fdir_ustat_add: 0
fdir_ustat_remove: 0
fdir_fstat_add: 0
fdir_fstat_remove: 0
fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 32764
fdir_coll: 0
fdir_match: 948      <-- netperf TCP_RR
fdir_miss: 529004675
fdir_ustat_add: 4
fdir_ustat_remove: 0
fdir_fstat_add: 44
fdir_fstat_remove: 0
Transaction latency using flow separation
Experiment 1 results

- Baseline (no background traffic) gives 30000 transactions per second
- With background traffic using RSS over all cores gives increase in transaction latency reducing transactions per second to ~5000
- The RSS patch (don't forward traffic on core 0) brings the transaction latency back to (almost) the same case as the baseline
- In all cases the control traffic is bound to core 0
Experiment 2: Flow separation in-line traffic

- Inline control within bulk data (on same incoming interface)
- Study latency of TCP transactions
- Work in progress
Results in-line

Transaction latency wo/w RSS path
Flow mix and 64 byte packets

Zoom in of 64 byte packets
Classifier small packet problem

Seems we drop a lot packets before they are classified

DCB (Data Center Bridging) has a lot of features to prioritize different type of traffic. But only for IEEE 802.1Q

VMDq2 suggested by Peter Waskiewicz Jr at Intel
Investigate hardware limits by transmitting as much as possible from all cores simultaneously.
### pktgen/setup

<table>
<thead>
<tr>
<th>Interface</th>
<th>eth0</th>
<th>eth1</th>
<th>eth2</th>
<th>eth3</th>
<th>eth4</th>
<th>eth5</th>
<th>eth6</th>
<th>eth7</th>
<th>eth8</th>
<th>eth9</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU-core</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>Mem node</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

eth4, eth5 on x4 slot
Setup

CPU0 Quad-core
QPI
QPI
IOH Tylersburg

CPU1 Quad-core
QPI
QPI
IOH Tylersburg

Memory node 0

eth0
eth1
eth2
eth3

Memory node 1

eth6
eth7
eth8
eth9
eth4
eth5
TX w. 10 * 10g ports
93Gb/s “Optimal”

10 * 10G tx performance w. 1500 byte pkts
Shortest route

Gbit/s
0 2 4 6 8 10
eth0 eth1 eth2 eth3 eth4 eth5 eth6 eth7 eth8 eth9

Interface
Conclusions

- We have shown traffic separation in a high-end multi-core PC with classifier NICs by assigning one CPU core as control and the other as forwarding cores. Our method:
  - Interrupt affinity to bind control traffic to core 0
  - Modified RSS to spread forwarding traffic over all except core 0
  - Modified the flow-director implementation slightly by only letting local (control) traffic populate the flowdir table.

- There are remaining issues with packet drops in in-line separation

- We have shown 93Gb/s simplex transmission bandwidth on a fully equipped PC platform
That's all

Questions?
Rwanda example
Lagos next

This document represents the topology of pilot phase of EKO-connect project based on wireless links.

KTH/CSD project 2010
Low-Power Development
Some ideas

Power consumption
SuperMicro X7SPA @ 16.5 Volt with picoPSU

<table>
<thead>
<tr>
<th>Watt</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.98</td>
<td>Power-Off</td>
</tr>
<tr>
<td>13.53</td>
<td>Idle</td>
</tr>
<tr>
<td>14.35</td>
<td>1 core</td>
</tr>
<tr>
<td>15.51</td>
<td>2 Core</td>
</tr>
<tr>
<td>15.84</td>
<td>3 Core</td>
</tr>
<tr>
<td>16.50</td>
<td>4 Core</td>
</tr>
</tbody>
</table>

Routing Performance about 500,000 packet/sec in optimal setup.
Example herjulf.se
14 Watt by 55Ah battery
bifrost/USB + lowpower disk
Running on battery
SuperCapacitors
DOM - Optical Monitoring

Optical modules can support optical link monitoring
RX, TX power, temperatuer, alarms etc

Newly added support to Bifrost/Linux
ethtool -D eth3


Temp: 25.5 C
Vcc: 3.28 V
Tx-Bias: 20.5 mA
TX-pwr: -3.4 dBm (0.46 mW)
RX-pwr: -15.9 dBm (0.03 mW)
Control and forwarding plane separation
on an open-source router

Linux Kongress
2010-09-23 in Nürnberg

Robert Olsson, Uppsala University
Olof Hagsand, KTH
More than 10 year in production at Uppsala University

UU-1

UU-2

AS 2834

DMZ

IPv4/IPv6

OSPF

Local BGP peering in Uppsala

Full Internet routing via EBGP/IBGP

2 * XEON 5630 TYAN 7025

4 *10g ixgbe sfp+ LR/SR

Now at 10g towards ISP, SFP+ 850 nm, 1310nm

ISP/SUNET AS1653

Stockholm

UU-Net

L-green

L-red

AS1653

AS2834

Internal UU-Net
Motivation

- Separate control-plane from forwarding plane
  A la IETF FORCES
- Control-plane: sshd, bgp, stats, etc on CPU core 0
- Forwarding-plane: Bulk forwarding on core1,...,coreN
- This leads to robustness of service against overload and DOS attacks, etc
- Enabled by:
  - multi-core CPUs
  - NIC hw classifiers
  - Fast Buses (QPI/PCI-E gen2)
Control-plane separation on a multi-core

Incoming traffic

Classifier

Control traffic

Forwarding traffic

Outgoing traffic

CE (core0)

FE1 (core1)

FE2 (core2)

... 

FEN (coreN)

Control-Element

Forwarding-Elements

Router
Hi-End Hardware

XEON 2 x E5630
TYAN S7025 Motherboard
Intel 82599
Block hardware structure

CPU0
Quad-core

DDR3

QPI

QPI

IOH
tylerburg

PCI-E Gen.2 x16
PCI-E Gen.2 x16
PCI-E Gen.2 x4

ESI

More I/O devices

CPU1
Quad-core

DDR3

QPI

QPI

IOH
tylerburg

PCI-E Gen.2 x16
PCI-E Gen.2 x16
PCI-E Gen.2 x8
Hi-End Hardware/Latency

![Graph showing latency results with stride 128 for different processors and configurations.](image)
Hardware - NIC

Intel 10g board Chipset 82599 with SFP+

Open chip specs. Thanks Intel!
Classification in the Intel 82599

The classification in the Intel 82599 consists of several steps, each is programmable. This includes:
- **RSS** (Receiver-side scaling): hashing of headers and load-balancing
- **N-tuples**: explicit packet header matches
- **Flow-director**: implicit matching of individual flows.
Routing daemons

Packet forwarding is done in Linux kernel
Routing protocols is run in user-space daemons

Currently tested versions of quagga
Bgp, OSPF both IPv4, IPv6
Cisco API
Experiment 1: flow separation external source

- Bulk forwarding data from source to sink (10Gb/s mixed packet lengths): mixed flow and packet lengths
- Netperf's TCP transactions emulated control data from a separate host
- Study latency of TCP transactions
N-tuple or Flowdirector

ethtool -K eth0 ntuple on

ethtool -U eth0 flow-type tcp4 src-ip 0x0a0a0a01 src-ip-mask 0xFFFFFFFF dst-ip 0 dst-ip-mask 0 src-port 0 src-port-mask 0 dst-port 0 dst-port-mask 0 vlan 0 vlan-mask 0 user-def 0 user-def-mask 0 action 0

ethtool -u eth0

N-tuple is supported by SUN Niu and Intel ixgbe driver.

Actions are: 1) queue 2) drop

But we were lazy and patched ixgbe for ssh and BGP to use CPU0
N-tuple or Flowdirector

Even more lazy... we found the flow-director was implicitly programmed by outgoing flows. So both incoming and outgoing would use the same queue.

So if we set affinity for BGP, sshd etc we could avoid the N-tuple filters

Example:
```
taskset -c 0 /usr/bin/sshd
```

Neat....
RSS is still using CPU0

So we both got our “selected traffic”
Plus the bulk traffic from RSS

We just want RSS to use “other” CPU's
Just a one-liner...

diff --git a/drivers/net/ixgbe/ixgbe_main.c b/drivers/net/ixgbe/ixgbe_main.c
index lb1419c..08bbd85 100644
--- a/drivers/net/ixgbe/ixgbe_main.c
+++ b/drivers/net/ixgbe/ixgbe_main.c
@@ -2379,10 +2379,10 @@ static void ixgbe_configure_rx(struct ixgbe_adapter *adapter)
   mrqc = ixgbe_setup_mrqc(adapter);

   if (adapter->flags & IXGBE_FLAG_RSS_ENABLED) {
-    /* Fill out redirection table */
-    for (i = 0, j = 0; i < 128; i++, j++) {
-      /* Fill out redirection table but skip index 0 */
-      for (i = 0, j = 1; i < 128; i++, j++) {
+      /* Fill out redirection table but skip index 0 */
+      for (i = 0, j = 1; i < 128; i++, j++) {
          if (j == adapter->ring_feature[RING_F_RSS].indices)
            j = 0;
+        j = 1;
          /* reta = 4-byte sliding window of
           * 0x00..(indices-1)(indices-1)00..etc. */
          reta = (reta << 8) | (j * 0x11);
### Patching RSS

<table>
<thead>
<tr>
<th>CPU-core</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of packets</td>
<td>0</td>
<td>196830</td>
<td>200860</td>
<td>186922</td>
<td>191866</td>
<td>186876</td>
<td>190106</td>
<td>190412</td>
</tr>
</tbody>
</table>

No traffic to CPU core 0 still RSS gives fairness between other cores
Transaction Performance
netperf TCP_RR

On “router”
taskset -c 0 netserver
Don't let forwarded packets program the flowdirector

A new one-liner patch....

@@ -5555,6 +5555,11 @@ static void ixgbe_atr(struct ixgbe_adapter *adapter, struct
       u32 src_ipv4_addr, dst_ipv4_addr;
       u8 l4type = 0;
+      if(!skb->sk) {
+          /* ignore nonlocal traffic */
+          return;
+      }
+      /* check if we're UDP or TCP */
+      if (iph->protocol == IPPROTO_TCP) {
+          th = tcp_hdr(skb);
Instrumenting the flow-director

ethtool -S eth0 | grep fdir
Flow-director stats/1

fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 8191
fdir_coll: 0
fdir_match: 195
fdir_miss: 573632813  <--- Bulk forwarded data from RSS
fdir_ustat_add: 1   <--- Old ssh session
fdir_ustat_remove: 0
fdir_fstat_add: 6
fdir_fstat_remove: 0
fdir_maxlen: 0

ustat → user stats
fstat → failed stats
Flow-director stats/2

fdir_maxhash: 0
fdir_free: 8190
fdir_coll: 0
fdir_match: 196
fdir_miss: 630653401
fdir_ustat_add: 2  <--- New ssh session
fdir_ustat_remove: 0
fdir_fstat_add: 6
fdir_fstat_remove: 0
Flow-director stats/3

- fdir_maxlen: 0
- fdir_maxhash: 0
- fdir_free: 8190
- fdir_coll: 0
- fdir_match: 206  <- ssh packets are matched
- fdir_miss: 645067311
- fdir_ustat_add: 2
- fdir_ustat_remove: 0
- fdir_fstat_add: 6
- fdir_fstat_remove: 0
Flow-director stats/4

fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 32768  <-- Now increased 32k
fdir_coll: 0
fdir_match: 0
fdir_miss: 196502463
fdir ust at_add: 0
fdir ust at_remove: 0
fdir_fstat_add: 0
fdir_fstat_remove: 0
Flow-director stats/5

fdir_maxlen: 0
fdir_maxhash: 0
fdir_free: 32764
fdir_coll: 0
fdir_match: 948  <-- netperf TCP_RR
fdir_miss: 529004675
fdir_ustat_add: 4
fdir_ustat_remove: 0
fdir_fstat_add: 44
fdir_fstat_remove: 0
Transaction latency using flow separation

![Transaction latency graph showing performance on QPUO]

- Baseline
- Load w. RSS
- Load w. RSS removed

TPD Transaction Performance on QPUO
Experiment 1 results

- Baseline (no background traffic) gives 30000 transactions per second
- With background traffic using RSS over all cores gives increase in transaction latency reducing transactions per second to ~5000
- The RSS patch (don’t forward traffic on core 0) brings the transaction latency back to (almost) the same case as the baseline
- In all cases the control traffic is bound to core 0
Experiment 2: Flow separation in-line traffic

- Inline control within bulk data (on same incoming interface)
- Study latency of TCP transactions
- Work in progress
Results in-line

Transaction latency wo/w RSS path
Flow mix and 64 byte packets

Zoom in of 64 byte packets
Classifier small packet problem

Seems we drop a lot packets before they are classified

DCB (Data Center Bridging) has a lot of features to prioritize different type of traffic. But only for IEEE 802.1Q

VMDq2 suggested by Peter Waskiewicz Jr at Intel
Experiment 3: Transmit limits

Investigate hardware limits by transmitting as much as possible from all cores simultaneously.
# pktgen/setup

<table>
<thead>
<tr>
<th>Interface</th>
<th>eth0</th>
<th>eth1</th>
<th>eth2</th>
<th>eth3</th>
<th>eth4</th>
<th>eth5</th>
<th>eth6</th>
<th>eth7</th>
<th>eth8</th>
<th>eth9</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU-core</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>Mem node</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**eth4, eth5 on x4 slot**
Setup

- CPU0: Quad-core
- CPU1: Quad-core
- Memory node 0
- IOH Tylersburg
- Memory node 1
- eth0
- eth1
- eth2
- eth3
- eth4
- eth5
- eth6
- eth7
- eth8
- eth9
TX w. 10 * 10g ports
93Gb/s “Optimal”
Conclusions

- We have shown traffic separation in a high-end multi-core PC with classifier NICs by assigning one CPU core as control and the other as forwarding cores. Our method:
  - Interrupt affinity to bind control traffic to core 0
  - Modified RSS to spread forwarding traffic over all except core 0
  - Modified the flow-director implementation slightly by only letting local (control) traffic populate the flowdir table.

- There are remaining issues with packet drops in in-line separation
- We have shown 93Gb/s simplex transmission bandwidth on a fully equipped PC platform
That's all

Questions?
Rwanda example
Low-Power Development
Some ideas

Power consumption
SuperMicro X7SPA @ 16.5 Volt with picoPSU

<table>
<thead>
<tr>
<th>Watt</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.98</td>
<td>Power-Off</td>
</tr>
<tr>
<td>13.53</td>
<td>Idle</td>
</tr>
<tr>
<td>14.35</td>
<td>1 core</td>
</tr>
<tr>
<td>15.51</td>
<td>2 Core</td>
</tr>
<tr>
<td>15.84</td>
<td>3 Core</td>
</tr>
<tr>
<td>16.50</td>
<td>4 Core</td>
</tr>
</tbody>
</table>

Routing Performance about 500.000 packet/sec in optimal setup.
Example herjulf.se
14 Watt by 55Ah battery
bifrost/USB + lowpower disk
Running on battery
SuperCapacitors
Optical modules can support optical link monitoring
RX, TX power, temperatures, alarms etc

Newly added support to Bifrost/Linux
DOM

```bash
ethtool -D eth3
```


- Temp: 25.5 C
- Vcc: 3.28 V
- Tx-Bias: 20.5 mA
- TX-pwr: -3.4 dBm (0.46 mW)
- RX-pwr: -15.9 dBm (0.03 mW)