A way towards Lower Latency and Jitter

Jesse Brandenburg
jesse.brandenburg@intel.com
BIO

• Jesse Brandeburg <jesse.brandeburg@intel.com>
  − A senior Linux developer in the Intel LAN Access Division, producing the Intel Ethernet product lines
  − Has been with Intel since 1994, and has worked on the Linux e100, e1000, e1000e, igb, ixgb, ixgbe drivers since 2002
  − Jesse splits his time between solving customer issues, performance tuning Intel's drivers, and bleeding edge development for the Linux networking stack
Acknowledgements

• Contributors
  − Anil Vasudevan, Eric Geisler, Mike Polehn, Jason Neighbors, Alexander Duyck, Arun Ilango, Yadong Li, Eliezer Tamir
“The speed of light sucks.”
- John Carmack
Current State

• NAPI is pretty good, but optimized for throughput
• Certain customers want extremely low end to end latency
  – Cloud providers
  – High Performance Computing (HPC)
  – Financial Services Industry (FSI)
• The race to the lowest latency has sparked user-space stacks
  – Most bypass the kernel stack
  – Examples include OpenOnload® application acceleration, Mellanox Messaging Accelerator (VMA), RoCEE/IBoE, RDMA/iWarp, and others [1]

[1] see notes for links to above products
Problem Statement

• Latency is high by default (especially for Ethernet)
• Jitter is unpredictable by default

Software Causes
• Scheduling/context switching of the process
• Interrupt balancing algorithms
• Interrupt rate settings
• Path length from receive to transmit

Hardware Causes
• # of fetches from memory
• Latency inside the network controller
• Interrupt propagation
• Power Management (NIC, PCIe, CPU)
# Latency and Jitter Contributors

<table>
<thead>
<tr>
<th>Key sources today</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Hardware Latency</td>
<td>New Hardware</td>
</tr>
<tr>
<td>Software Execution Latency</td>
<td>Opportunity</td>
</tr>
<tr>
<td>Scheduling / Context Switching</td>
<td>Opportunity</td>
</tr>
<tr>
<td>Interrupt Rebalancing</td>
<td>Interrupt-to-core mapping</td>
</tr>
<tr>
<td>Interrupt Moderation/Limiting</td>
<td>Minimize/Disable throttling (ITR=0)</td>
</tr>
<tr>
<td>Power Management</td>
<td>Disable (or limit) CPU power management, PCIe power management</td>
</tr>
<tr>
<td>Bus Utilization (jitter)</td>
<td>Isolate device</td>
</tr>
</tbody>
</table>
## Latency and Jitter Contributors

<table>
<thead>
<tr>
<th>Key sources today</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw Hardware Latency</td>
<td>New Hardware</td>
</tr>
<tr>
<td><strong>Software Execution Latency</strong></td>
<td><strong>Opportunity</strong></td>
</tr>
<tr>
<td><strong>Scheduling / Context Switching</strong></td>
<td><strong>Opportunity</strong></td>
</tr>
<tr>
<td>Interrupt Rebalancing</td>
<td>Interrupt-to-core mapping</td>
</tr>
<tr>
<td>Interrupt Moderation/Limiting</td>
<td>Minimize/Disable throttling (ITR=0)</td>
</tr>
<tr>
<td>Power Management</td>
<td>Disable (or limit) CPU power management, PCIe power management</td>
</tr>
<tr>
<td>Bus Utilization (jitter)</td>
<td>Isolate device</td>
</tr>
</tbody>
</table>
Traditional Transaction Flow

1. App transmits thru sockets API
   - Passed down to driver and h/w unblocked
   - TX is “Fire and Forget”

2. App checks for receive

3. No immediate receive – thus block

4. Packet received & Interrupt generated
   - Interrupt subject to Int Rate & Int Balancing

5. Driver passes to Protocol

6. Protocol/Sockets wakes App

7. App received data thru sockets API

8. Repeat

Very inefficient for low-latency traffic
Latency Breakdown 2.6.36
Latency Breakdown kernel v3.5

v3.5 Round Trip Packet Timings

- Total: 5722 ns
Jitter Measurements
min/max in us measured by netperf

![Graph showing Jitter Measurements]

- udp_rr
- arx-off-1
- 3.9.15

- tcp_rr
- arx-off-1
Jitter Measurements
standard deviation measured by netperf

Stddev_latency netperf

- udp_rr
  - arx-off-1
  - 3.9.15

- tcp_rr
  - arx-off-1
  - 3.9.15
Proposed Solution

• Improve the software latency and jitter by driving the receive from user context

• Result
  − The Low Latency Sockets proof of concept
Low Latency Sockets (LLS)

• LLS is a software initiative to reduce networking latency and jitter within the kernel
• Native protocol stack is enhanced with a low latency path in conjunction with packet classification (queue picking) by the NIC
• Transparent to applications and benefits those sensitive to unpredictable latency
• Top down busy-wait polling replaces interrupts for incoming packets
New Low-Latency Transaction Flow

1. App transmits thru sockets API
   • Passed down to driver and h/w unblocked
   • TX is “Fire and Forget”
2. App checks for data (receive)
3. Check device driver for pending packet (poll starts)
4. Meanwhile, packet received to NIC
5. Driver processes pending packet
   • Bypasses context switch & interrupt
6. Driver passes to Protocol
7. App receives data through sockets API
8. Repeat
Proof of Concept

• Code developed on 2.6.36.2 kernel
• Initial numbers done with ixgbe driver from out of tree
• Includes lots of timing and debug code
• Currently reliant upon
  – hardware flow steering
  – one queue pair (Tx/Rx) per CPU
  – Interrupt affinity configured
Proof of Concept Results (2.6.36.2)
Jitter Results
min/max latency in us, as measured by netperf

<table>
<thead>
<tr>
<th>Min_latency</th>
<th>Max_latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>udp_rr</td>
<td>tcp_rr</td>
</tr>
<tr>
<td>arx-off-1</td>
<td>arx-off-1</td>
</tr>
<tr>
<td>3.9.15</td>
<td>3.9.15</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>udp_rr</td>
<td>tcp_rr</td>
</tr>
<tr>
<td>arx-off-90</td>
<td>arx-off-90</td>
</tr>
<tr>
<td>lls-r03</td>
<td>lls-r03</td>
</tr>
</tbody>
</table>
Jitter Results
standard deviation as measured by netperf

Stddev_latency

- udp_rr
  - arx-off-1
  - 3.9.15
- tcp_rr
  - arx-off-1
  - 3.9.15
- udp_rr
  - arx-off-90
  - lls-r03
- tcp_rr
  - arx-off-90
  - lls-r03
Possible Issues

• Unpalatable structure modifications
  – struct sk_buff
  – struct sk

• Dependency on driver or kernel implemented flow steering

• Current amount of driver code to implement
  – Current work already in progress on a much simpler version

• Default enabled?
  – How can we turn this on and off
    – Don’t want a socket option – defeats the purpose

• Security issues?
  – Application can now force hardware/memory reads – unlikely to be an issue
  – The new poll runs in syscall context, which should be safe but we need to be careful to not create a new vulnerability
  – does this new implementation create other problems?
Current work

• Work in progress includes
  – Further simplified driver using a polling thread
  – Port of the current code to v3.5

• Future work
  – Post current v3.5 code to netdev (Q4 – 2012)
  – Design and refactor based on comments
  – Make sure new flow is measurable and debuggable
Code

• Git tree posted at:
  – https://github.com/jbrandeb/lls.git

• Branches
  – v2.6.36.2_lls
    – Original 2.6.36.2 based prototype
  – v3.5.1_lls
    – Port of code to v3.5.1 stable (all features may not work yet)
Contact

jesse.brandenburg@intel.com
e1000-devel@lists.sourceforge.net
netdev@vger.kernel.org
Summary

• Customers want a low latency and low jitter solution
  − We can make one native to the kernel
• LLS prototype shows a possible way forward
  − Achieved lower latency and jitter

• Discussion
  − What would you do differently?
  − Do you want to help?
Abstract

• Development-in-progress of a new in-kernel interface to allow applications to achieve lower network latency and jitter

• Creates a new driver interface to allow an application to drive a poll through the socket layer all the way down to the device driver

• Benefits are
  – applications do not have to change
  – Linux networking stack is not bypassed in any way
  – Minimized latency of data to the application
  – Much more predictable jitter

• The design, implementation and results from an early prototype will be shown, and current efforts to refine, refactor, and upstream the design will be discussed

• Affected areas include the core networking stack, and network drivers