Page-Flip Technology for use within the Networking Linux Stack

Authors:
Jesse Brandeburg
John Ronciak
Ganesh Venkatesan

Intel Corporation

Agenda

- Background and Problem Statement
- Solution Design and Implementation
- Hardware and System Assumptions
- Test Methodology
- Results
- Surprises
- Conclusions
- Current Issues with Implementation
- Next Steps

Background

- High Speed Networking Needs a Mechanism to Avoid Data Copies in the Kernel and Stack
- Device DMA and Kernel Space/User Space Boundary Copy
- Zero-Copy Mechanisms Avoid Kernel/User Copy
- DDP and RDMA are Zero-Copy mechanisms – Complicated and Cumbersome to Use

Background (cont.)

- Assumed that Page-Flip is Less Complicated and More Efficient
- Dislike of TOE and RDMA due to Complexity
  - Lack of Intra-Stack Access
- Dave Miller Discussions (netdev and lkml)
- FreeBSD Work on Page-Flip Mechanisms

Problem Statement

- Assumed Kernel and Stack Need a Zero-Copy Kernel-Space to User-Space Mechanism
- High Speed Networking Will Probably Require Performance Enhancements (i.e. 10 Gbps Ethernet) that Includes a Zero-Copy Mechanism

Implementation

- Decided on Using 2.6.4 Kernel
  - Better Memory Management Mechanism than 2.4
- Modified Driver to Split Protocol Headers from Payload Data
- Flag SKB to Indicate HW/Driver Prepared a Page-Flippable SKB
- Modified “skb_copy_datagram_iovec()” to Support New “flip_page_mapping()” Function
- Modify SKB Free Routine to support Frags with Null Addresses

Assumptions

- Temporary Measures Due To HW Limitations
  - Application Needs to Allocate Data Receive Area in 4KByte Multiples (i.e. PAGE_SIZE)
  - Area Must be PAGE_SIZE Aligned
  - Modified 'nttcp-1.4.7' to Use “vmalloc()” for Above Application Requirements
- MTU of 4KB or 8KB Used
- 2.6.4 Kernel Used 4KB Page Size
- No Debug Options Enabled

Platform Hardware

- Test Platform
  - Dual Processor 1.8 and 2.4 Ghz Pentium® 4
  - Hyper-Thread Technology Disabled
  - 512 Mbytes RAM
  - NIC Support of Split Header and Protocol Checksum Verification

Test Methodology

- Measured Performance of “Page-Flip” against “Copy-Once” (Current) Mechanisms
- Two Major Test Runs
  - Application Never Touched Data
  - Application Touched Data Forcing Data into Cache and Validating Received Data
- Each Instance of Test had 3 Runs with Results Averaged

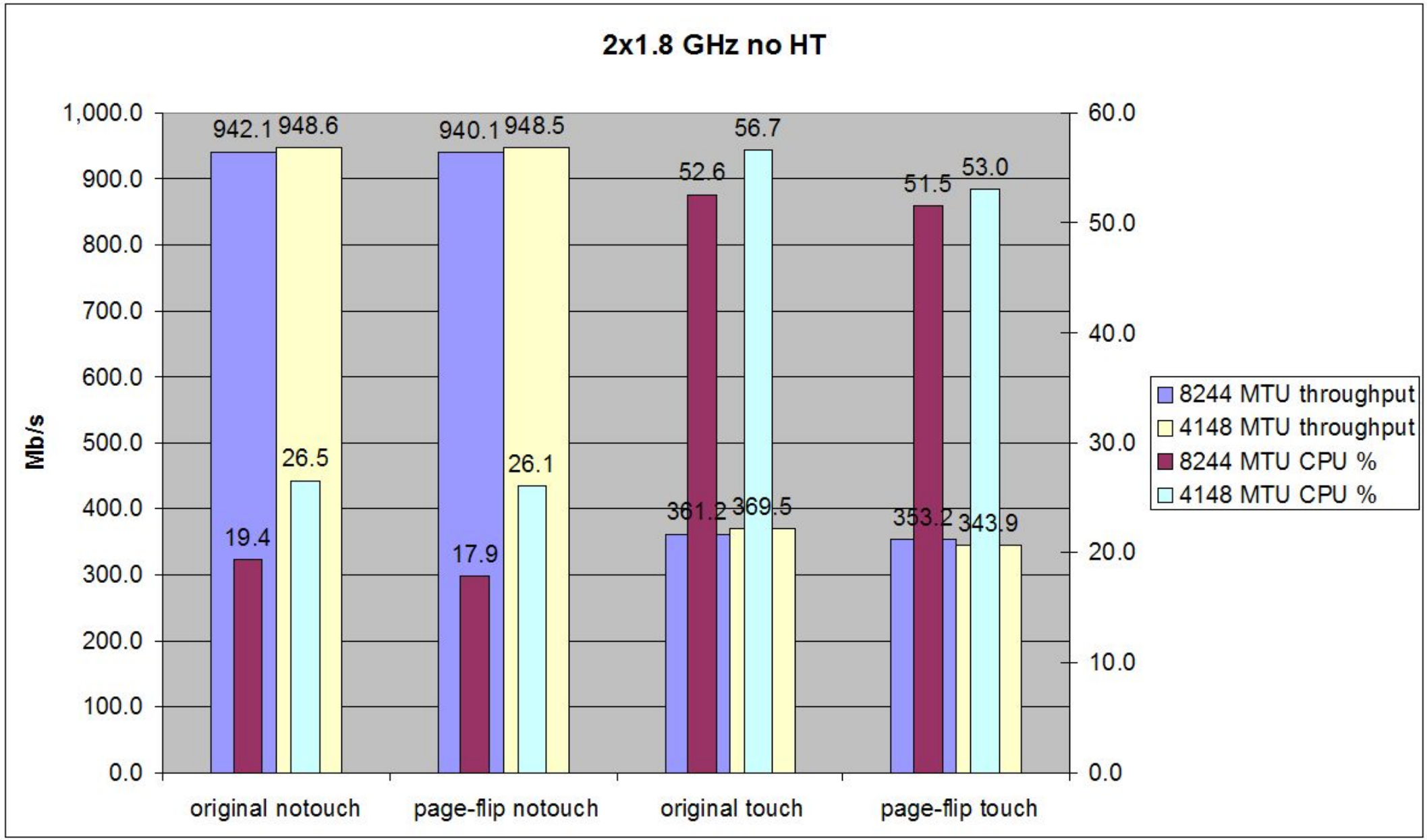
Test Methodology (cont.)

- Oprofile Used to Locate “Hot-Spots”
- CPU and Network Utilization Measured with 'sar'
  - Corrected Version of 'sar' Used to Measure CPU Utilization Correctly

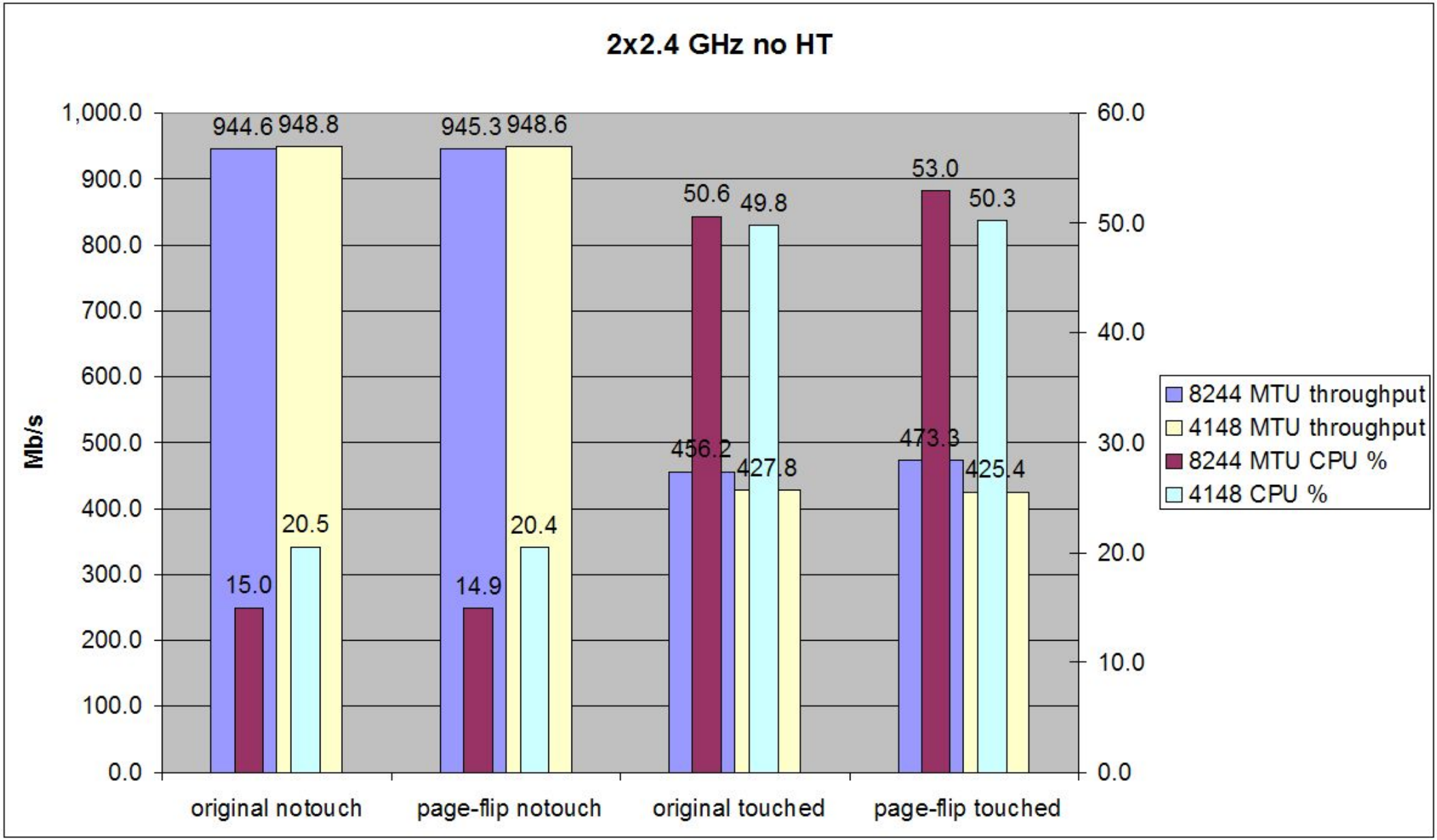
Results

- Graphs Show Resultant Data
- Touched Packet Data On Slower Processors Shows Slight CPU Reduction from Page-Flip Mechanism w/ Decrease in Throughput
- Efficiency (Mbits/CPU = efficiency) is Lower for All Cases Using Page-Flip

Results


2x1.8 GHz no HT
Mb/s
1,000.0
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
942.1
948.6
19.4
26.5
940.1
948.5
17.9
26.1
52.6
56.7
361.2
369.5
51.5
53.0
353.2
343.9
original notouch
page-flip notouch
original touch
page-flip touch
8244 MTU throughput
4148 MTU throughput
8244 MTU CPU %
4148 MTU CPU %

Results


2x2.4 GHz no HT
Mb/s
1,000.0
900.0
800.0
700.0
600.0
500.0
400.0
300.0
200.0
100.0
0.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
944.6
948.8
15.0
20.5
945.3
948.6
14.9
20.4
50.6
49.8
456.2
427.8
53.0
50.3
473.3
425.4
original notouch
page-flip notouch
original touched
page-flip touched
8244 MTU throughput
4148 MTU throughput
8244 MTU CPU %
4148 CPU %

Surprises

- Results Were Unexpected
- Expected Some Benefit from Page-Flip (Gain in Efficiency)
- Some Benefit from Copy Mechanism (Cache Warming w/ Packet Data) Which Data Confirms

Oprofile Results

- Locks Associated with Page-Flip Code Accounts for Majority of Stalls
- Stalls Associated with TLB (Translation Look-aside Buffer) Flush is Very Painful

Conclusions

- Page-Flip Mechanism Offers Little (or no) Performance Enhancements for Zero-Copy Receive Due To:
  - Cache Issues (Obvious from Touched Data Results)
  - Heavy Cost to Prepare and Complete Page-Flip
- Possible Use Could Still be Embedded Environments (or Slower Processor Environments)
- Page-Flip Won't Scale with Processor Speed

Current Issues

- Lack of Commercial NIC HW that Supports Header Splitting
- Lack of Any HW that is Ideal (Dave Miller Ideal Version) which Supplies Flow Identification
- Current Code has a Bug when 'clone_skb()' is Used (Freeing Issue of Cloned SKB)
- Assumptions Made to Enable this Testing Limits it's Usefulness Outside Academic Use

Next Steps

- Do We Try to Optimize Page-Flip Code Path?
  - VM Locking Issues
  - Optimize TLB Flushes
- What About Application API Changes?
  - What's needed here? receive_file() API?
- Look to Other Zero-Copy Mechanisms?

Contact Information

- John Ronciak - john.ronciak@intel.com
- Jesse Brandeburg - jesse.brandeburg@intel.com
- Ganesh Venkatesan - ganesh.venkatesan@intel.com

- Code for Kernel Patch and Driver Mods are Available from our Sourceforge Site: sf.net/projects/e1000