Packet-based Network Traffic Monitoring and Analysis with GPUs

Wenji Wu, Phil DeMar
wenji@fnal.gov, demar@fnal.gov

GPU Technology Conference 2014
March 24-27, 2014 SAN JOSE, CALIFORNIA

Background

- Main uses for network traffic monitoring & analysis tools:
  - Operations & management
  - Capacity planning
  - Performance troubleshooting
- Levels of network traffic monitoring & analysis:
  - Device counter level (snmp data)
  - Traffic flow level (flow data)
  - Packet level (The Focus of this work)
    - security analysis
    - application performance analysis
    - traffic characterization studies

Background (cont.)

Characteristics of packet-based network monitoring & analysis applications

- Time constraints on packet processing.
- Compute and I/O throughput-intensive
- High levels of data parallelism.
  - Packet parallelism. Each packet can be processed independently
  - Flow parallelism. Each flow can be processed independently
- Extremely poor temporal locality for data
  - Typically, data processed once in sequence; rarely reused

The Problem

Packet-based traffic monitoring & analysis tools face performance & scalability challenges within high-performance networks.

- High-performance networks:
  - 40GE/100GE link technologies
  - Servers are 10GE-connected by default
  - 400GE backbone links & 40GE host connections loom on the horizon.
- Millions of packets generated & transmitted per sec

Monitoring & Analysis Tool Platforms (I)

- Requirements on computing platform for high performance network monitoring & analysis applications:
  - High Compute power
  - Ample memory bandwidth
  - Capability of handing data parallelism inherent with network data
  - Easy programmability

Monitoring & Analysis Tool Platforms (II)

- Three types of computing platforms:
  - NPU/ASIC
  - CPU
  - GPU

Features	NPU/ASIC	CPU	GPU
High compute power	Varies	✖	✔
High memory bandwidth	Varies	✖	✔
Easy programmability	✖	✔	✔
Data-parallel execution model	✖	✖	✔

Architecture Comparison

Our Solution

Use GPU-based Traffic Monitoring & Analysis Tools

Highlights of our work:

- Demonstrated GPUs can significantly accelerate network traffic monitoring & analysis
  - 11 million+ pkts/s without drops (single Nvidia M2070)
- A generic I/O architecture to move network traffic from wire into GPU domain
- Implemented a GPU-accelerated library for network traffic capturing, monitoring, and analysis.
  - Dozens of CUDA kernels, which can be combined in a variety of ways to perform monitoring and analysis tasks

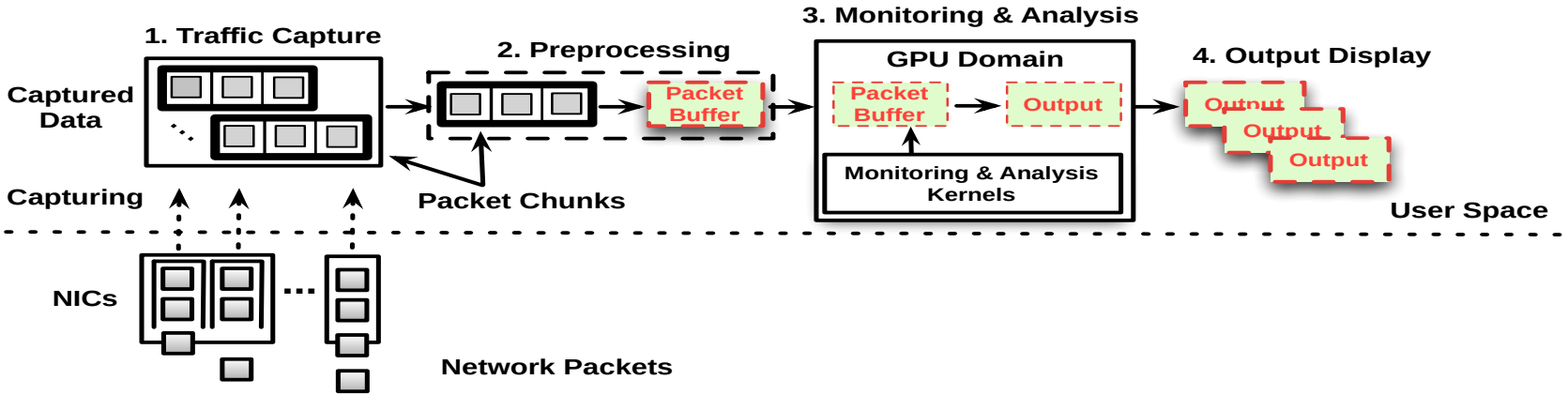
Key Technical Issues

- GPU's relatively small memory size:
  - Nvidia M2070 has 6 GB Memory, K10 has 8 GB Memory
  - Workarounds:
    - Mapping host memory into GPU with zero-copy technique? ✖
    - Partial packet capture approach ✔
- Need to capture & move packets from wire into GPU domain without packet loss
- Need to design data structures that are efficient for both CPU and GPU

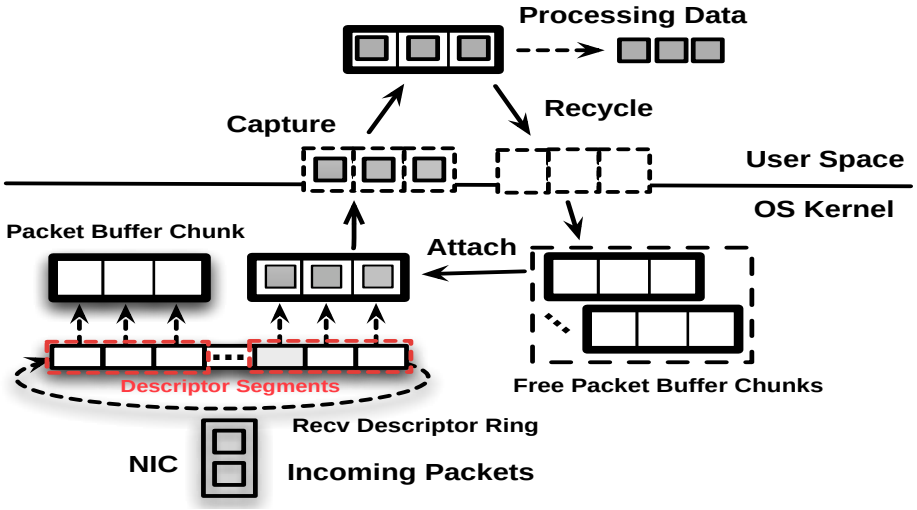
System Architecture

Four Types of Logical Entities:

- Traffic Capture
- Preprocessing
- Monitoring & Analysis
- Output Display



Packet I/O Engine



Goal:

- **To avoid packet capture loss**

Key techniques

- **A novel ring-buffer-pool mechanism**
- **Pre-allocated large packet buffers**
- **Packet-level batch processing**
- **Memory mapping based zero-copy**

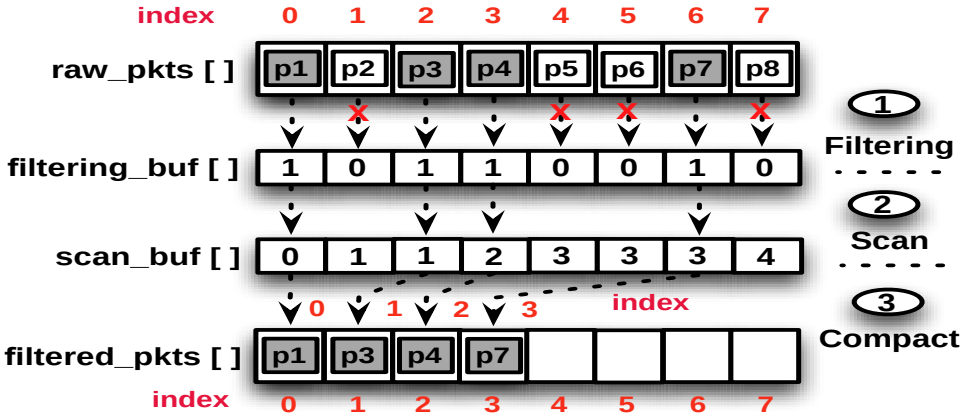
Key Operations

- **Open**
- **Capture**
- **Recycle**
- **Close**

GPU-based Network Traffic Monitoring & Analysis Algorithms

- A GPU-accelerated library for network traffic capturing, monitoring, and analysis apps.
  - Dozens of CUDA kernels
  - Can be combined in a variety of ways to perform intended monitoring & analysis operations

Packet-Filtering Kernel



We use Berkeley Packet Filter (BPF) as the packet filter

A few basic GPU operations, such as sort, prefix-sum, and compact.

Advanced packet filtering capabilities at wire speed are necessary so that we only analyze those packets of interest to us.

Sample Use Case -1

Using our GPU-accelerated library, we developed a sample use case to monitor network status:

- Monitor networks at different levels:
  - from aggregate of entire network down to one node
- Monitor network traffic by protocol
- Monitor network traffic information per node:
  - Determine who is sending/receiving the most traffic
  - For both local and remote addresses
- Monitor IP conversations:
  - Characterizing by volume, or other traits.

Packet Parallelism

Sample Use Case 1 – Data Structures

Three key data structures were created at GPU:

- *protocol_stat[]*
  - an array that is used to store protocol statistics for network traffic, with each entry associated with a specific protocol.
- *ip_snd[]* and *ip_rcv[]*
  - arrays that are used to store traffic statistics for IP conversations in the send and receive directions, respectively.
- *ip_table*
  - a hash table that is used to keep track of network traffic information of each IP address node.

These data structures are designed to reference themselves and each other with relative offsets such as array indexes.

Sample Use Case 1 – Algorithm

1. Call Packet-filtering kernel to filter packets of interest
2. Call Unique-IP-addresses kernel to obtain IP addresses
3. Build the ip_table with a parallel hashing algorithm
4. Collect traffic statistics for each protocol and each IP node
5. Call Traffic-Aggregation kernel to build IP conversations

Sample use case 2

Using our GPU-accelerated library, we developed a sample use case to monitor bulk data transfer:

- Monitor bulk data transfer rate
  - from aggregate of entire network down to one node
- Analyze data transfer performance bottlenecks
  - Sender-limited, network-limited, or receiver-limited?
  - Network-limited?
    - Packet drops, or packet reordering?

Packet Parallelism

+

Flow Parallelism

Sample Use Case 2 – Algorithm

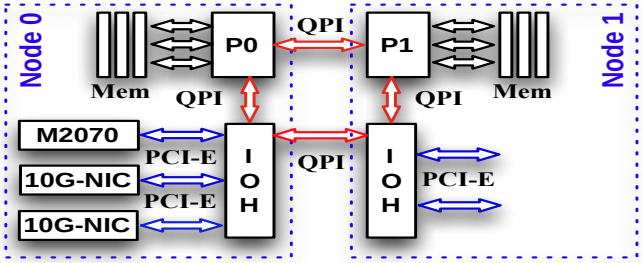
Monitoring Mode:

1. Call Packet-filtering kernel to filter packets of interest
2. Call TCP-throughput kernel to calculate data transfer rates

Analyzing Mode:

1. Call Packet-filtering kernel to filter packets of interest
2. Classify packets into flows
3. Call TCP-analysis kernel to analyze performance bottlenecks

Prototyped System

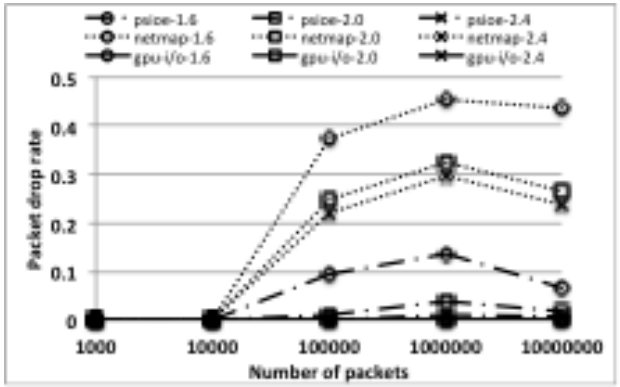


Prototyped System

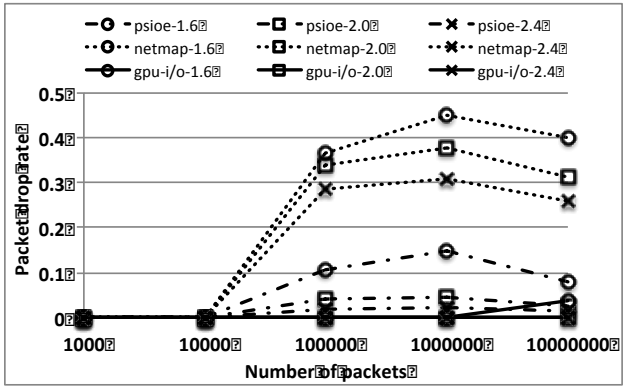
- Our application is developed on Linux.
- CUDA 5.0 programming environment.
- The packet I/O engine is implemented on Intel 82599-based 10GigE NIC

- A two-node NUMA system
- Two 8-core 2.67GHz Intel X5650 processors.
- Two Intel 82599-based 10GigE NICs
- One Nvidia M2070 GPU.

Performance Evaluation
Packet I/O Engine



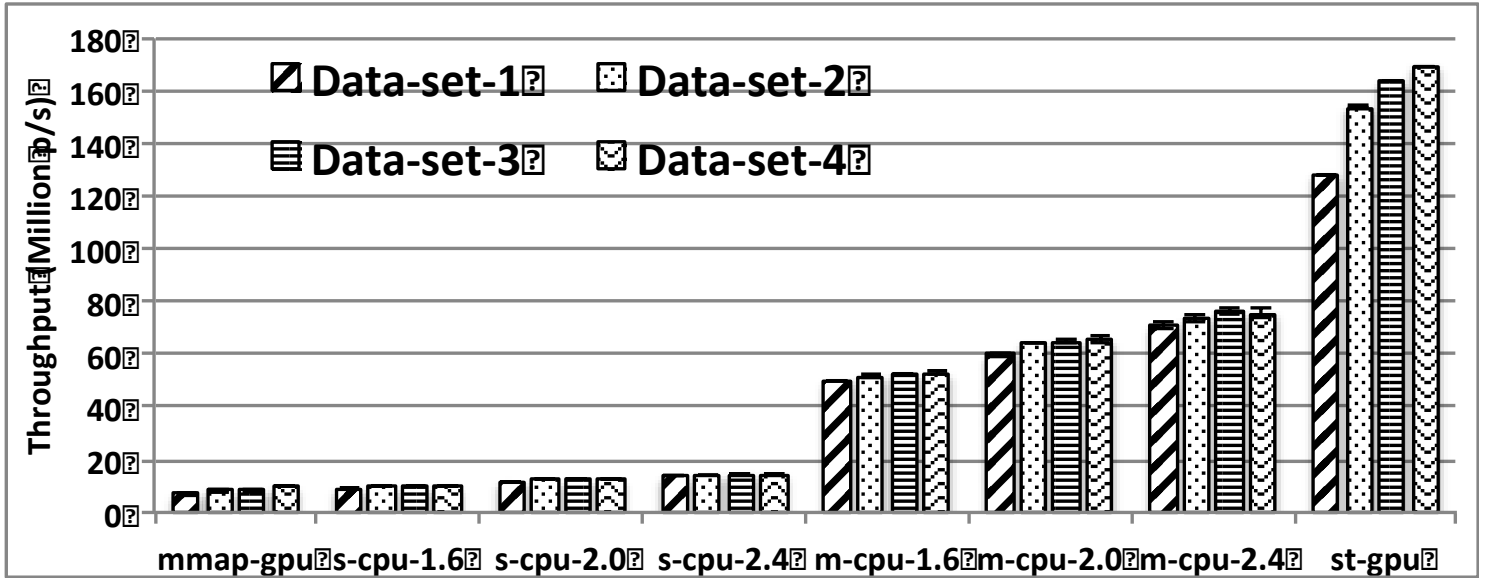
Packet drop rate – *one-nic-exp*



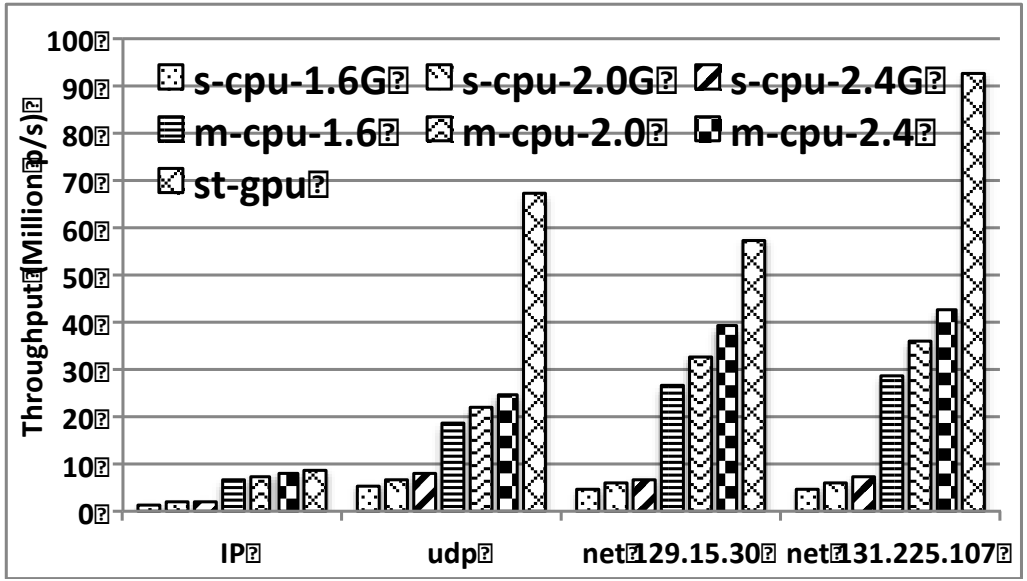
Packet drop rate – *two-nic-exp*

- Our Packet I/O engine (GPU-I/O) achieves better performance than NETMAP and PSIOE in avoiding packet capture loss.

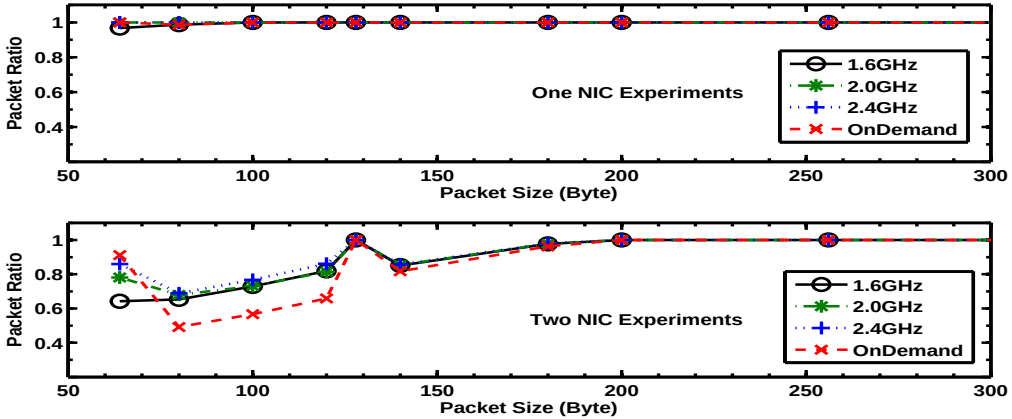
Performance Evaluation
GPU-based Packet Filtering Algorithm



Performance Evaluation
GPU-based Sample Use Case - 1



Performance Evaluation – Overall System Performance



In the experiments:

- Generators transmitted packets at wire rate.
- Packet sizes are varied across the experiments.
- Prototype system run in full operation with sample use case 1.

Key Take Aways

- GPUs accelerate network traffic monitoring & analysis.
- A generic I/O architecture to move network traffic from wire into GPU domain.
- A GPU-accelerated library for network traffic capturing, monitoring, and analysis.
  - Dozens of CUDA kernels, which can be combined in various ways to perform monitoring and analysis tasks.

Question?

Thank You!

Email: wenji@fnal.gov and demar@fnal.gov