Maxeler Apps
Line Rate Packet Capture

MAXELER
Technologies
MAXIMUM PERFORMANCE COMPUTING

Dec 2014

Line Rate Packet Capture

Problem

With the speed and bandwidth of networks increasing as more services are transitioning online it becomes highly important to understand what data is being transferred across a network.

Current packet capture solutions are lossy and tend to make sacrifices on what data is logged due to these increasing demands by:

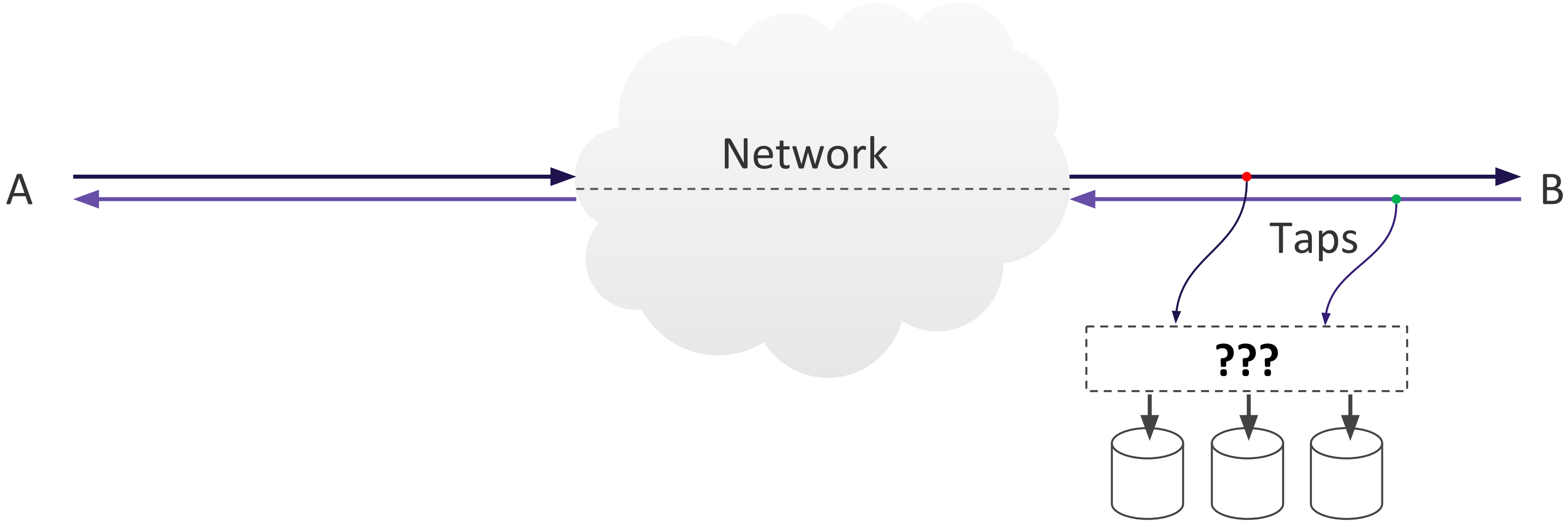
- Filtering traffic on a pre-set criteria
- Sampling a subset of data
- Putting short limits on retention

Existing Software Solutions: WireShark, tcpdump, pcap, ...

Line Rate Packet Capture

Logging **all** data allows

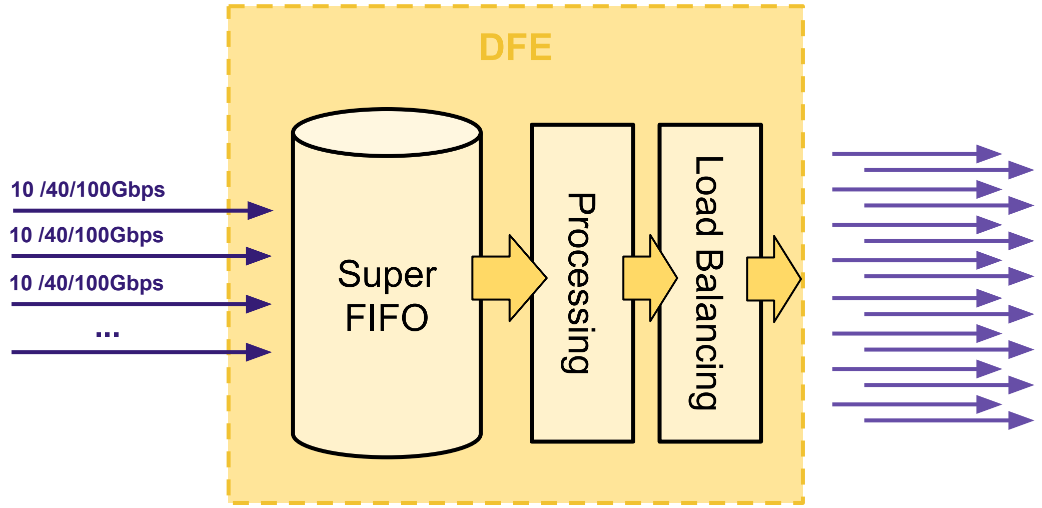
- Retroactive problem solving
- Logging/protecting against cyber threats
- Policy enforcement
- Debugging protocols/services
- Understanding how users use your network



Line Rate Packet Capture

Solution Overview

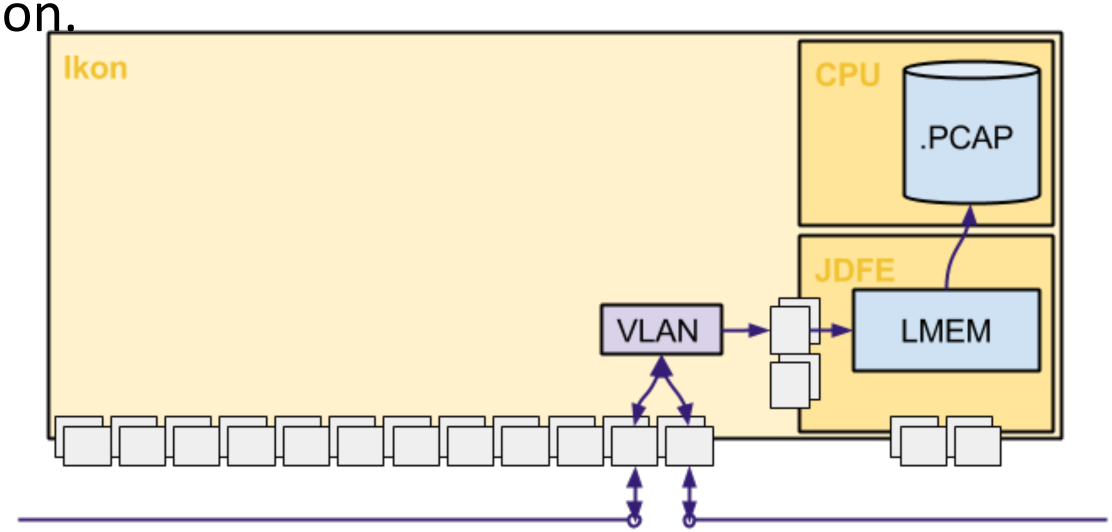
Use a DFE to buffer **all** network data at line rate, process, and pass off to CPU or cluster of storage backends



Packet Capture Implementation

Ikon Switch Overview

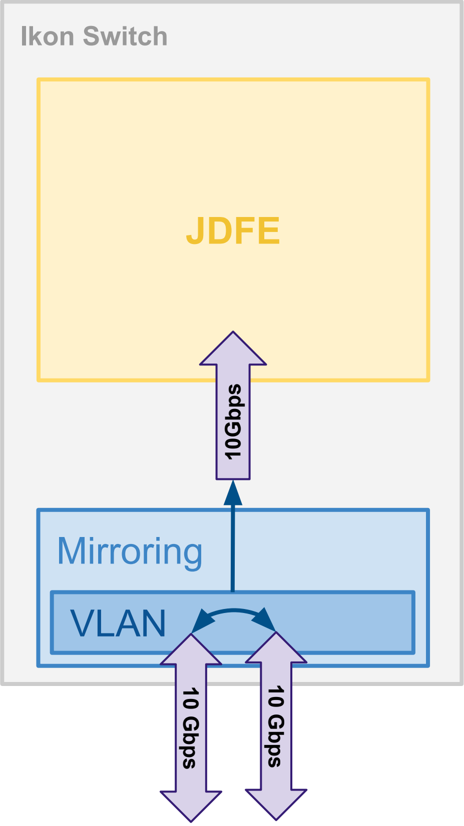
In this implementation user-defined ports on an Ikon switch are routed to the JDFE for buffering. For local capture the CPU transfers network data from the JDFE to a pcap file for later analysis. For distributed capture the JDFE sends data to a pool of capture servers for retention.



The JDFE's 192Gb LMEM is used as a buffer to allow bursts of up to ~20s of lossless 10Gbps capture from a single port.

Packet Capture Implementation

Ikon Switch Configuration



VLAN Setup

```
vlans {
  v_capture {
    vlan-id 2;
  }
}

interfaces {
  xe-0/0/20 {
    unit 0 {
      family ethernet-switching {
        vlan {
          members v_capture;
        }
      }
    }
  }
  xe-0/0/22 {
    unit 0 {
      family ethernet-switching {
        vlan {
          members v_capture;
        }
      }
    }
  }
}
```

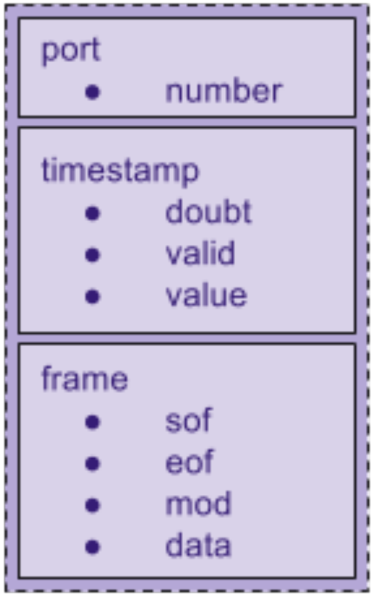
Mirroring Setup

```
forwarding-options {
  analyzer {
    a_capture {
      input {
        ingress {
          vlan v_capture;
        }
      }
      output {
        interface xe-0/0/32.0;
      }
    }
  }
}
```

Packet Capture Implementation

Data Types

capture data



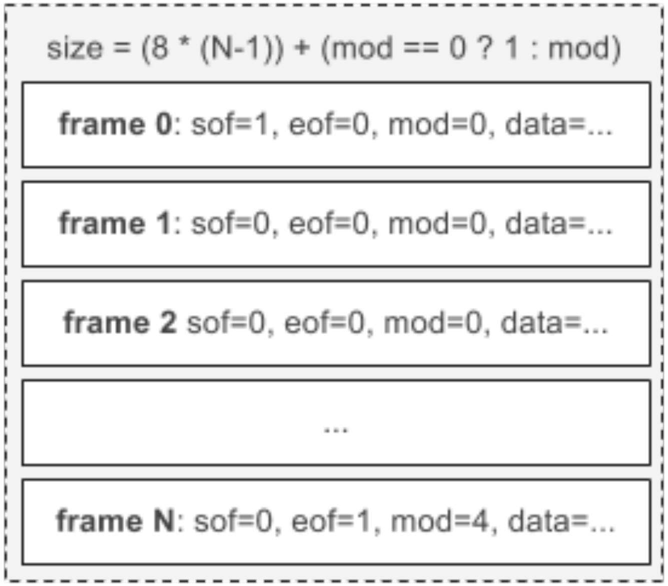
port
- number

timestamp
- doubt
- valid
- value

frame
- sof
- eof
- mod
- data

Capture data contains port, timestamp, and frame related info

packet



size = (8 * (N-1)) + (mod == 0 ? 1 : mod)

frame 0: sof=1, eof=0, mod=0, data=...

frame 1: sof=0, eof=0, mod=0, data=...

frame 2 sof=0, eof=0, mod=0, data=...

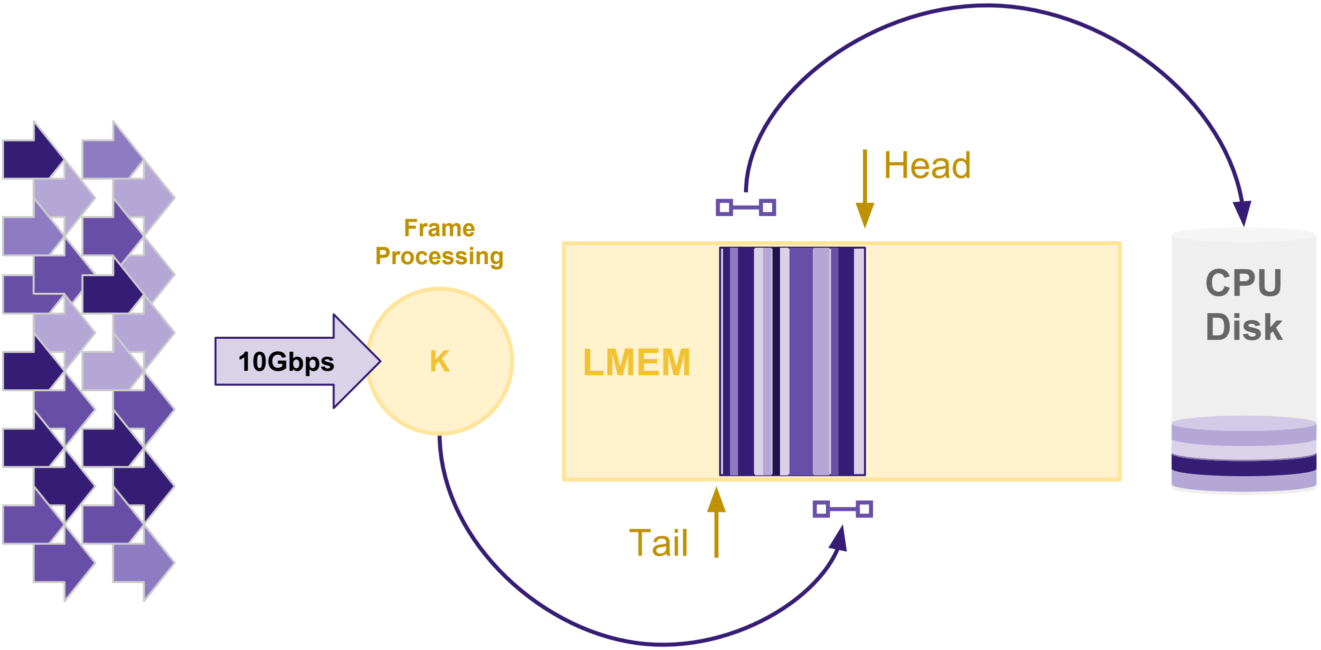
...

frame N: sof=0, eof=1, mod=4, data=...

Packets are composed of frames which contain start of frame, end of frame, size(data) % 8, and data fields

Packet Capture Implementation

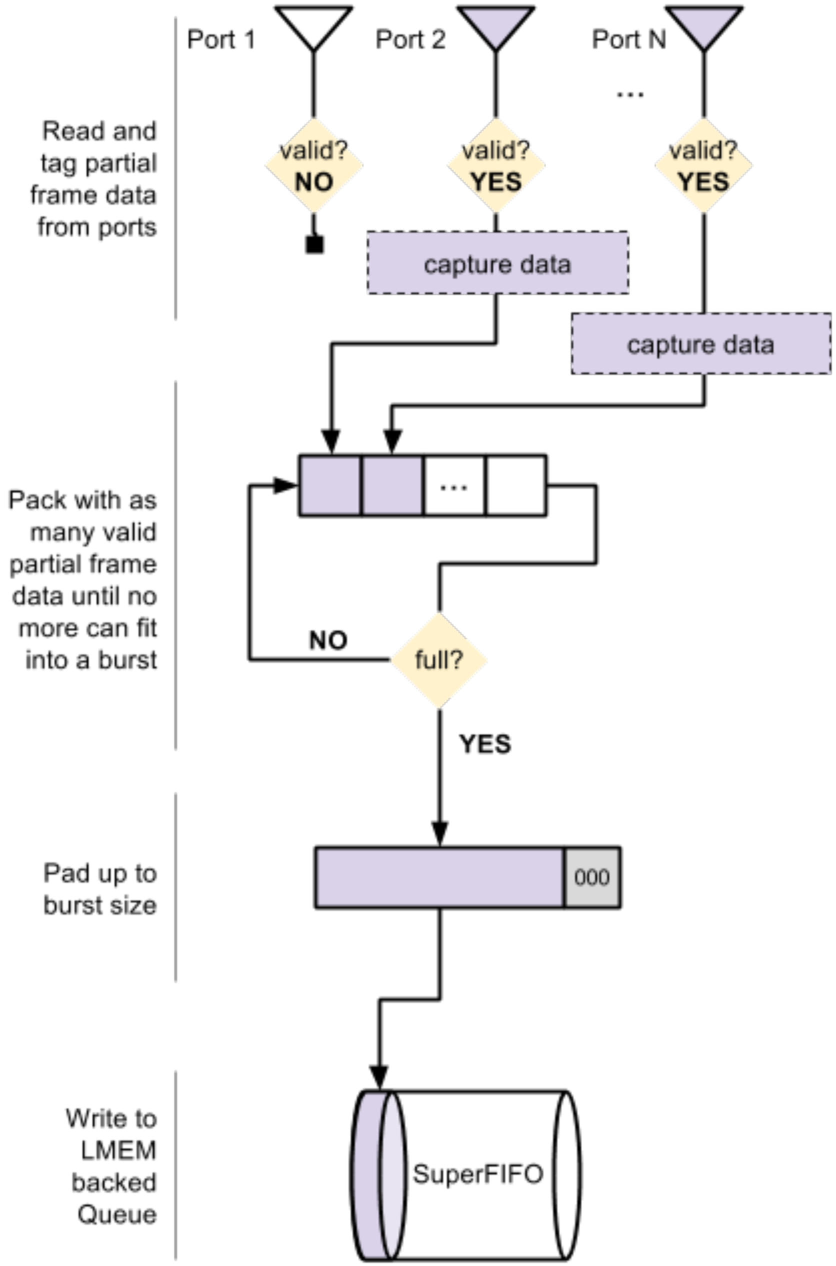
DFE Configuration Overview: Local Capture



CPU Service Loop

1. Wait for data
2. Read chunk of partial frames from LMEM
3. Reassemble frames
4. Write data to pcap file
5. Repeat

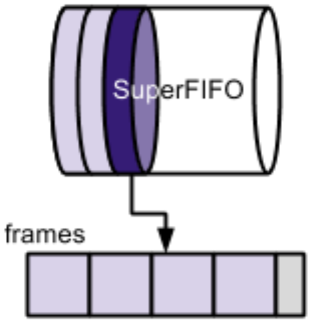
DFE Implementation


Port 1
Port 2
...
Port N
Read and tag partial frame data from ports
valid? NO
valid? YES
valid? YES
capture data
capture data
Pack with as many valid partial frame data until no more can fit into a burst
...
NO
full?
YES
Pad up to burst size
000
Write to LMEM backed Queue
SuperFIFO

CPU Implementation

Read from LMEM backed Queue

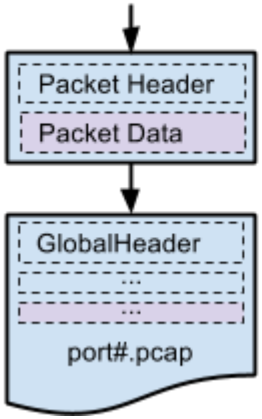
```
PacketCapture_read(frames, BURST_SIZE)
```



Reassemble packet data for each port

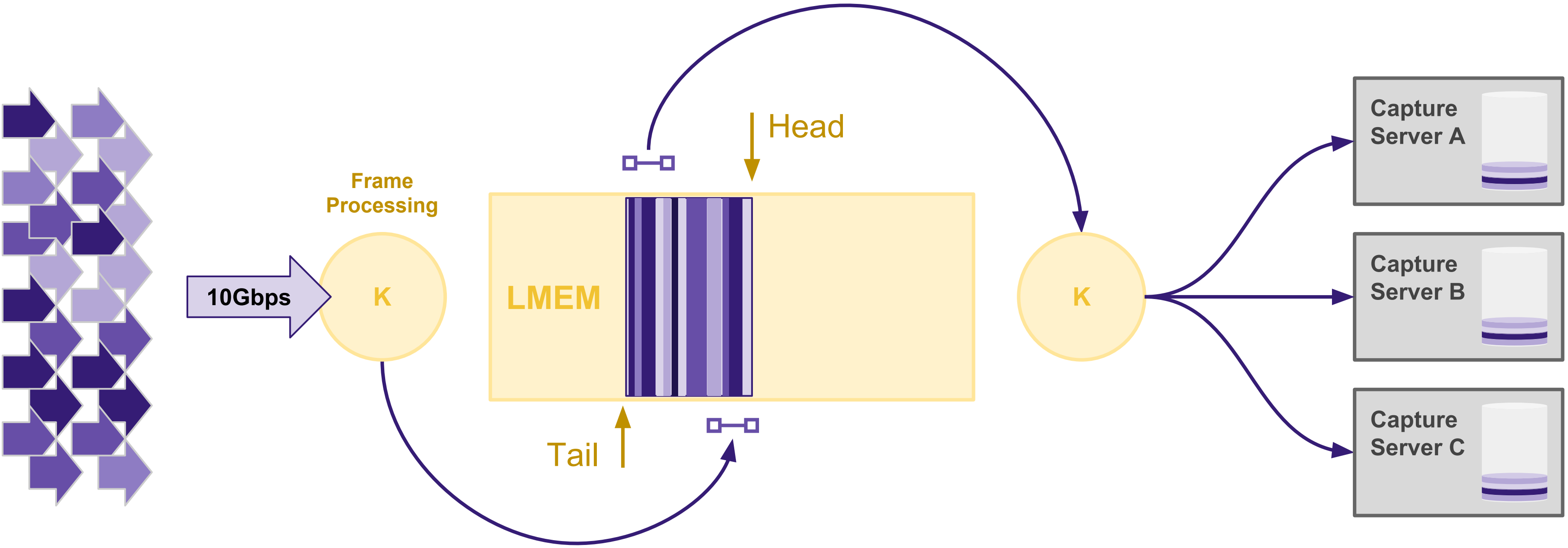
```
packets = []
for cd in capture_datas:
    port = cd.frame.port
    size = (cd.frame.mod == 0) ? 8 : cd.frame.mod
    data = cd.frame.data[0:(size * 8)]
    if frame.sof:
        packets[port] = data
        timestamps[port] = cd.timestamp
    else:
        packets[port].append(data)
    if frame.eof:
        packet = packets[port]
        timestamp = timestamps[port]
        pcap_write_packet(file, timestamp, packet)
```

Write packet to pcap file

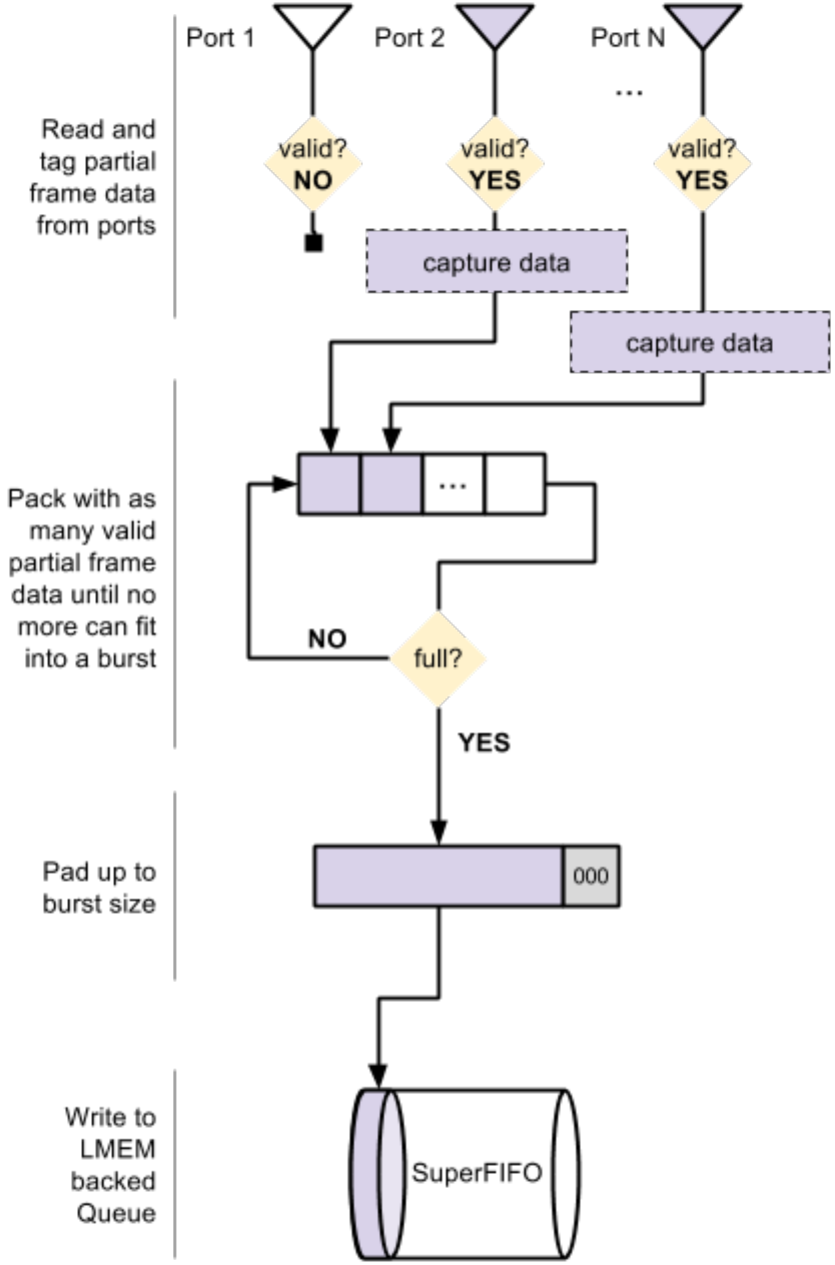


Packet Capture Implementation

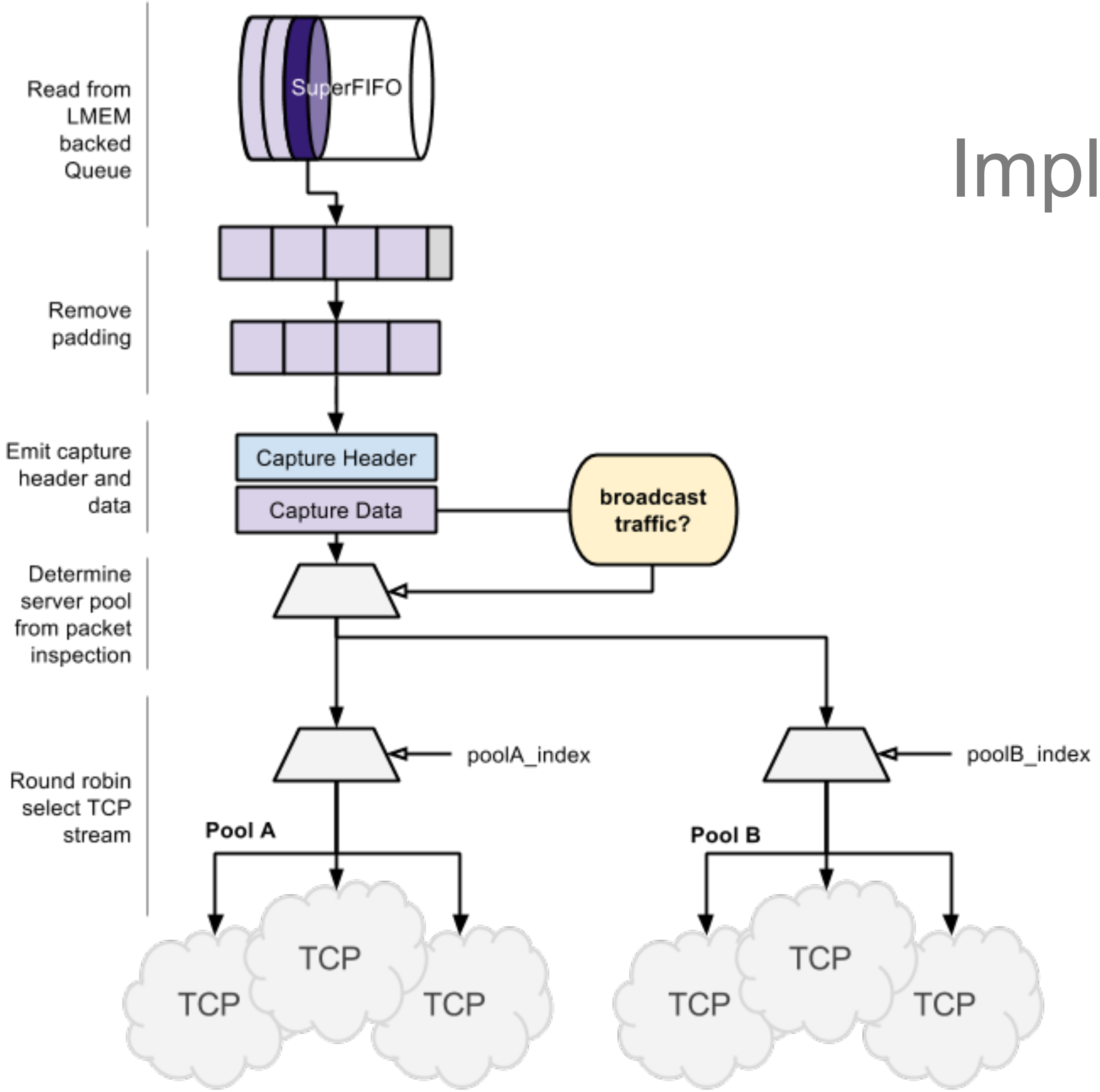
DFE Configuration Overview: Load Balanced Capture



DFE Implementation

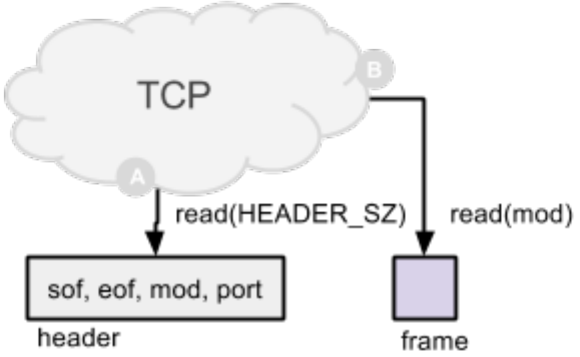

Port 1
Port 2
Port N
...
Read and tag partial frame data from ports
valid? NO
valid? YES
valid? YES
capture data
capture data
Pack with as many valid partial frame data until no more can fit into a burst
...
NO
full?
YES
Pad up to burst size
000
Write to LMEM backed Queue
SuperFIFO

DFE Implementation


Read from LMEM backed Queue
SuperFIFO
Remove padding
Emit capture header and data
Capture Header
Capture Data
broadcast traffic?
Determine server pool from packet inspection
Round robin select TCP stream
poolA_index
poolB_index
Pool A
Pool B
TCP
TCP
TCP
TCP
TCP
TCP

Capture Server Implementation

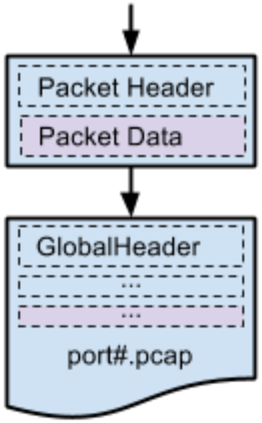
Receive header and packet data from TCP stream



Reassemble packet data for each port

```
packets = []
while(running)
  A read_capture_header(cd)
  B read_capture_data(cd)
    port = cd.frame.port
    data = cd.frame.data[0:(size * 8)]
    if frame.sof:
        packets[port] = data
        timestamps[port] = cd.timestamp
    else:
        packets[port].append(data)
    if frame.eof:
        packet = packets[port]
        timestamp = timestamps[port]
        pcap_write_packet(file, timestamp, packet)
```

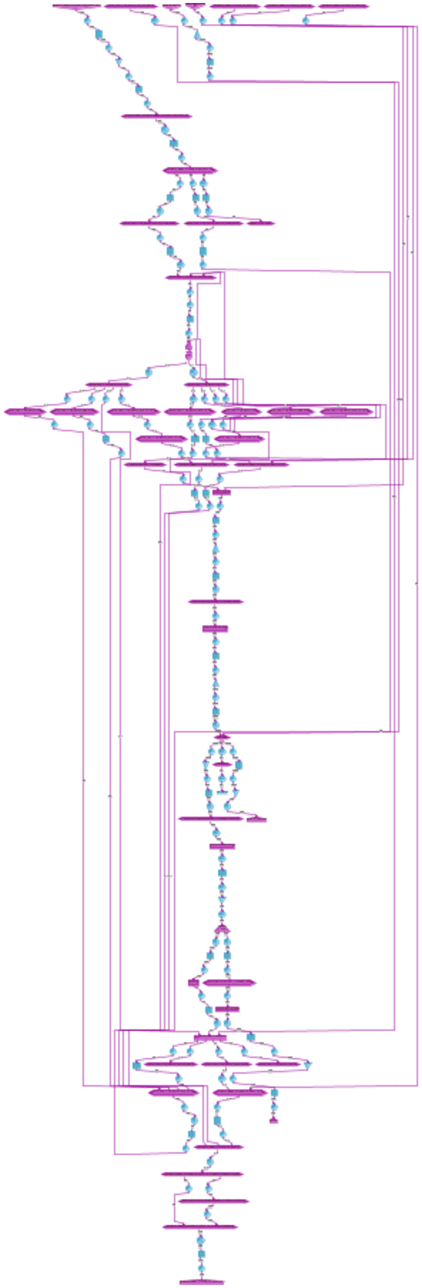
Write packet to pcap file



Resource Utilization & Manager Graph

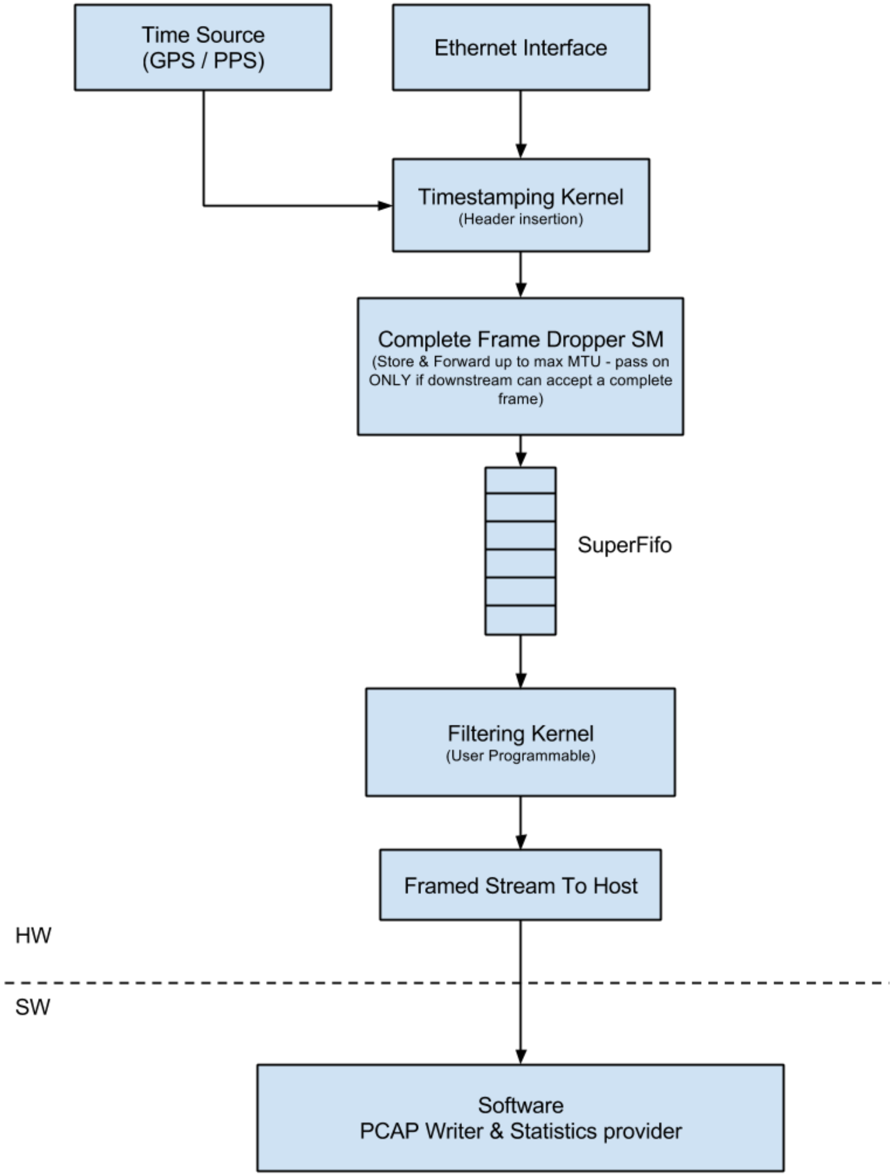
Single Port Capture

Type	Usage Absolute	Usage Percent
Logic utilization	49107 / 359200	13.67%
Primary FFs	83492 / 718400	11.62%
Secondary FFs	3937 / 718400	0.55%
Multipliers (18x18)	0 / 704	0.00%
DSP blocks	0 / 352	0.00%
Block memory (M20K)	347 / 2640	13.14%



Advanced Impl.

- High-Precision Timestamps
- Filtering
- Decoding
- Compression
- User Defined Behavior
- Stream to Storage System
- Lossless Packet Capture



Time Source
(GPS / PPS)
Ethernet Interface
Timestamping Kernel
(Header insertion)
Complete Frame Dropper SM
(Store & Forward up to max MTU - pass on ONLY if downstream can accept a complete frame)
SuperFifo
Filtering Kernel
(User Programmable)
Framed Stream To Host
HW
SW
Software
PCAP Writer & Statistics provider