Understanding and Improving the Cost of Scaling Distributed Event Processing

Shoaib Akram, Manolis Marazakis, and Angelos Bilas

shbakram@ics.forth.gr

Foundation for Research and Technology – Hellas (FORTH)

Institute of Computer Science (ICS)

“Big Data” and Role of Event Processing

- Amount of data produced is increasing
  - Data doubling faster than Moore’s law [www.emc.com]
  - Mainly driven by web
- Data needs to be processed with low latency
  - Modern web applications
  - Real time analytics
  - Finance, fraud detection etc.
- Data produced by many sources can be seen as events
- Event processing has potential for the data center
  - IBM InfoSphere Streams is an example

Challenges

- Rich processing capabilities
  - Functionally equivalent tasks in real-time
- Scale *yet* simple and efficient
  - Low end-to-end latency
  - Low energy consumption

Challenges

- Rich processing capabilities
  - Functionally equivalent tasks in real-time
- Scale *yet* simple and efficient (focus of this work)
  - Low end-to-end latency
  - High network utilization
  - Low energy consumption

Motivation 1-Understand

- Sources of complexity when aiming for scale?
- **This work:** Detailed study of a real event processing stack
  - Event processing stack → Processing plus distribution
  - Flow of events intra-node and inter-node

Motivation 2-Quantify and Improve

- What is the cost and could it be improved?
- **This Work:** Quantify, improve and measure impact
  - Up to 200% improvement in throughput on thin nodes
  - Up to 5x improvement in throughput on fat nodes
  - Reduction in energy consumption and infrastructure cost

Outline

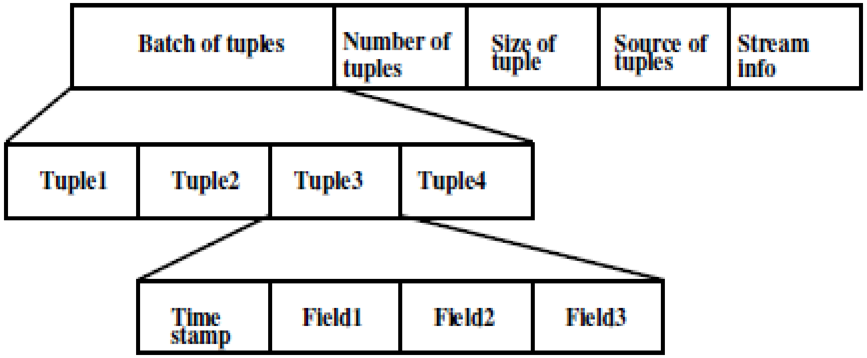
- ✓ Introduction and Motivation
- Stream Event Processing (Borealis)
- Optimizations
- Evaluation Methodology
- Results
  - Throughput
  - Energy
  - Projections for Future
- Conclusions

Stream Event Processing

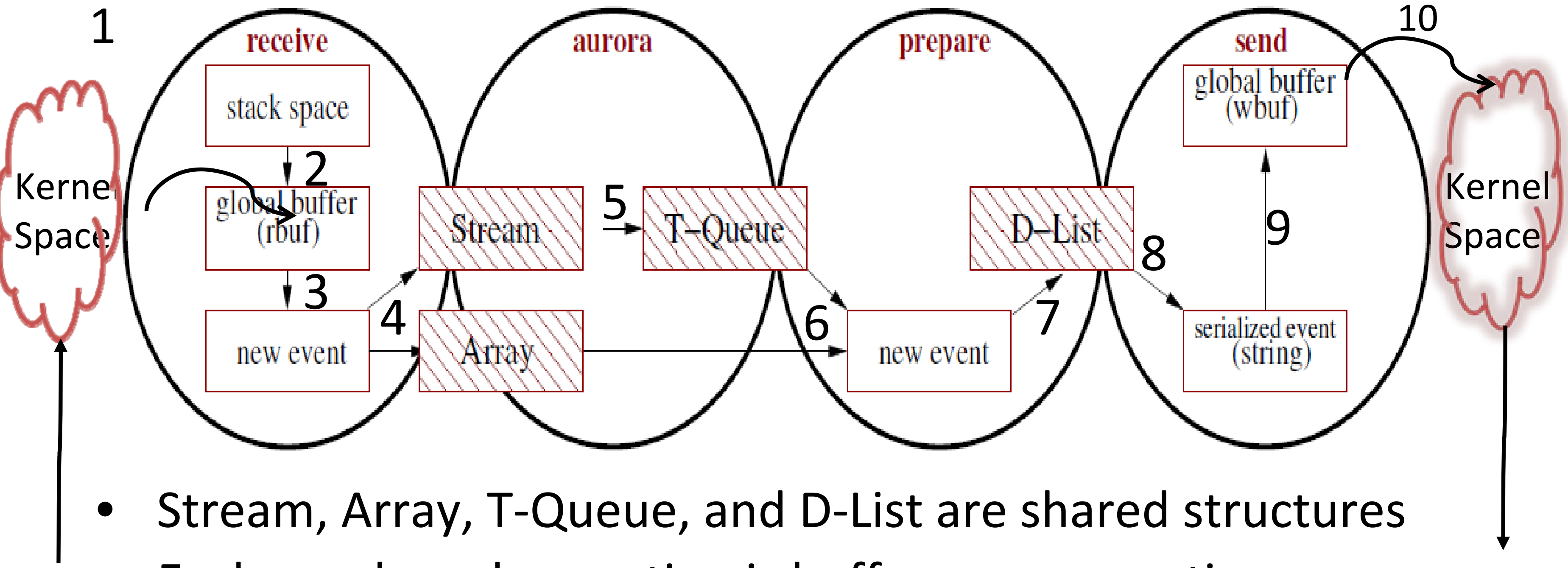
- Static queries and moving data
- Full set of database operators (filter, sort, union etc.)
- The end-user provides
  - Meaning of data in the stream (schema)
  - How to process data (query logic)
- Individual nodes run subset of query in a distributed setup
- Famous examples
  - Stream (Stanford)
  - System S (IBM)
  - Aurora/Borealis (MIT/Brown/Brandeis)

Events in Borealis

- Event contains tuples, info., and (optionally) arrays
- Events contain
  - Tuple has a time-stamp
  - and a number of fields
  - and arrays of data



End-to-end Datapath



- Stream, Array, T-Queue, and D-List are shared structures
- Each numbered operation is buffer copy operation
- After user-space optimizations, only 1, 4 and 10 remains
- Kernel space will also be bypassed (only 4 remains)

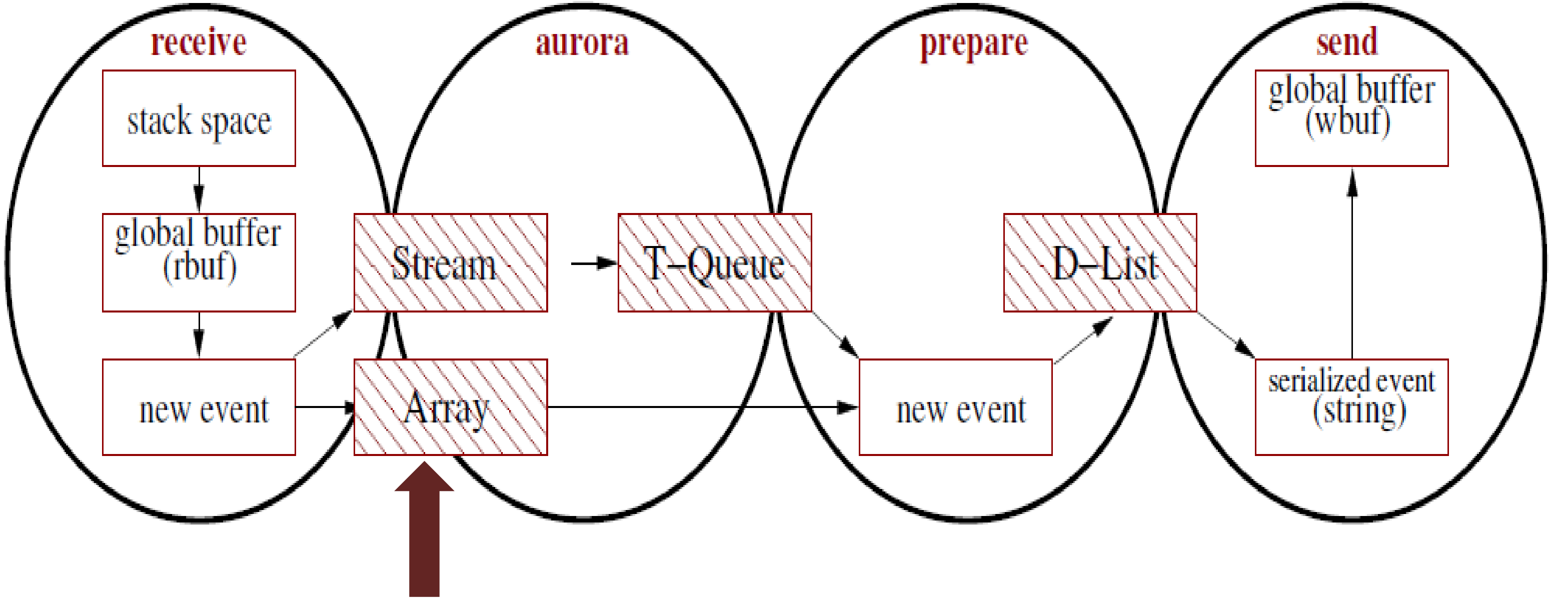
What makes these operations necessary?

- Well-defined interfaces
- Convenience
- Heterogeneity
- Portability
- Faults/Reliability
- Decoupling

Outline

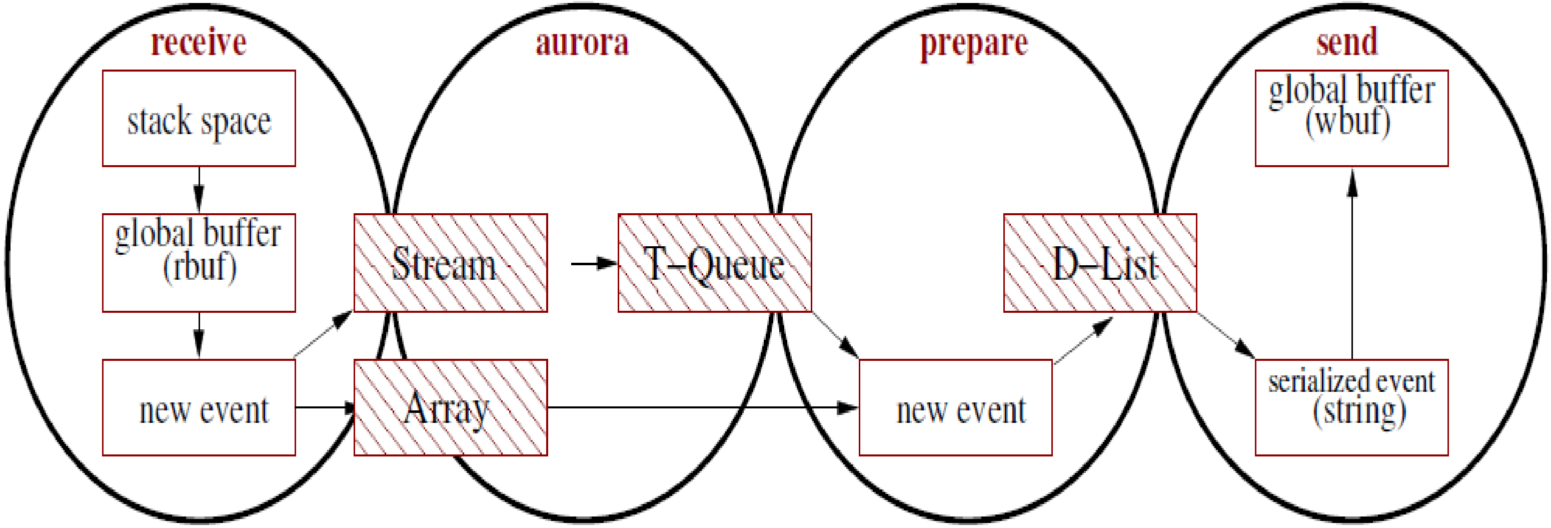
- ✓ Introduction and Motivation
- ✓ The Borealis Stream Processing Engine
- Optimizations
- Evaluation Methodology
- Results
  - Throughput
  - Efficiency
  - Projections for Future
- Conclusions

Flow Control



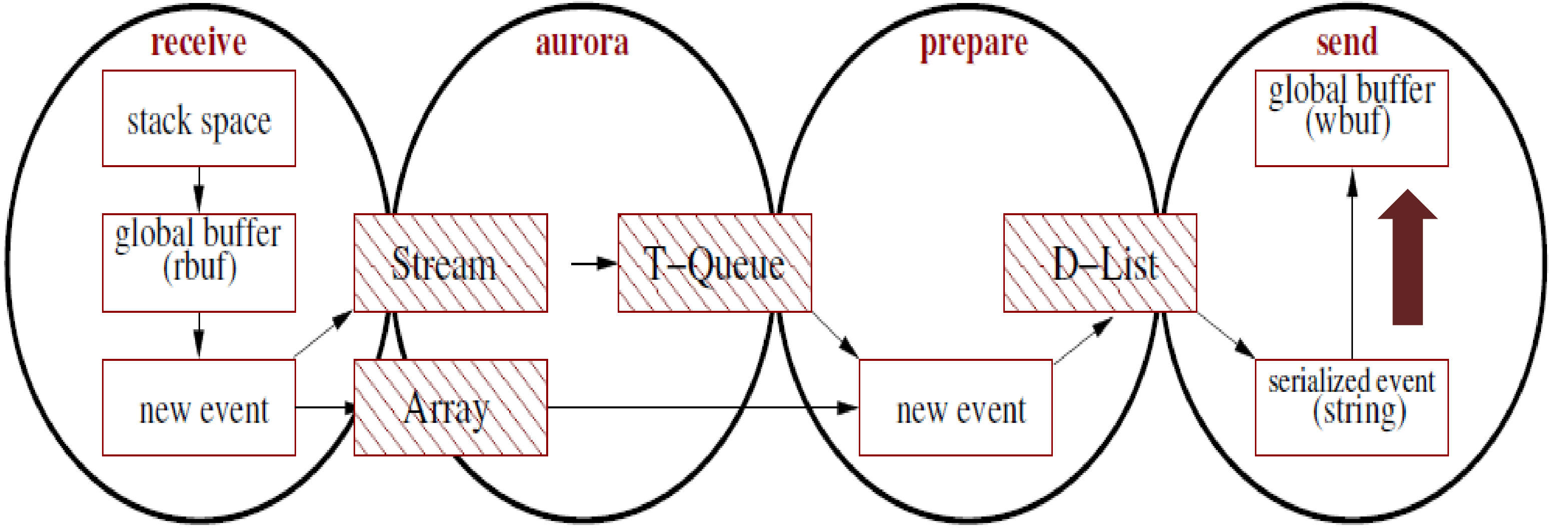
- No flow control in original Borealis (slow networks)
- Size of array is monitored for flow control

Message Queuing (Asynchronity)



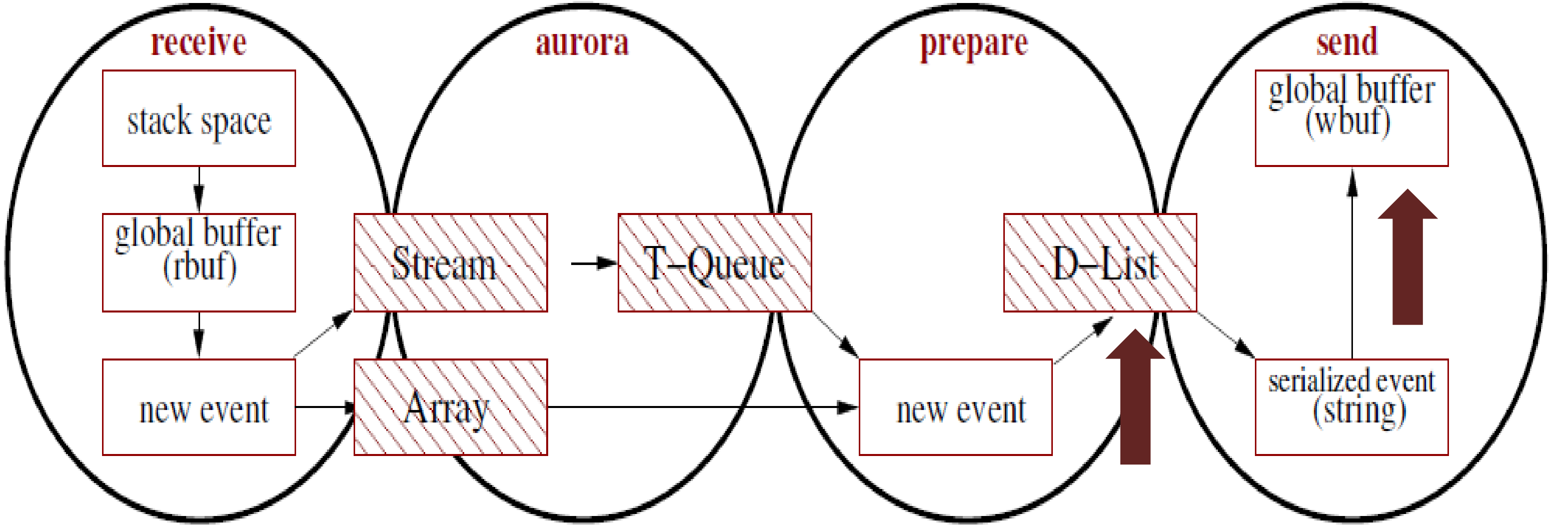
- Message queuing on the send path

Message Queuing (Asynchronity)



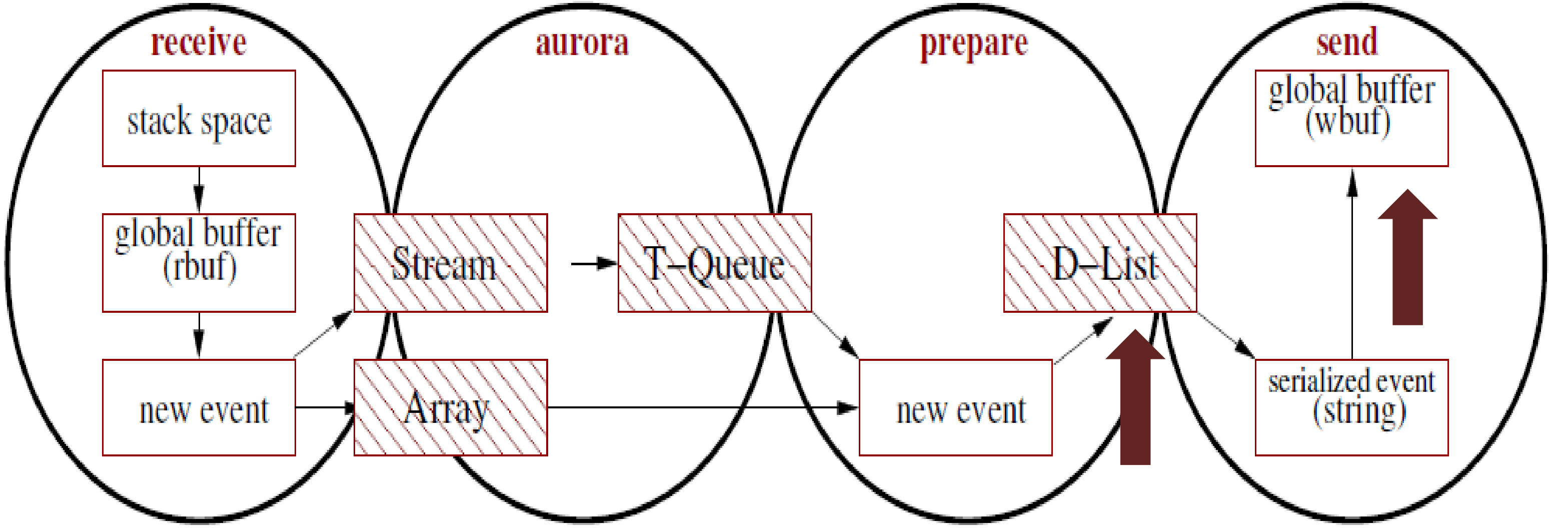
- Message queuing on the send path (1)wbuf

Message Queuing (Asynchronity)



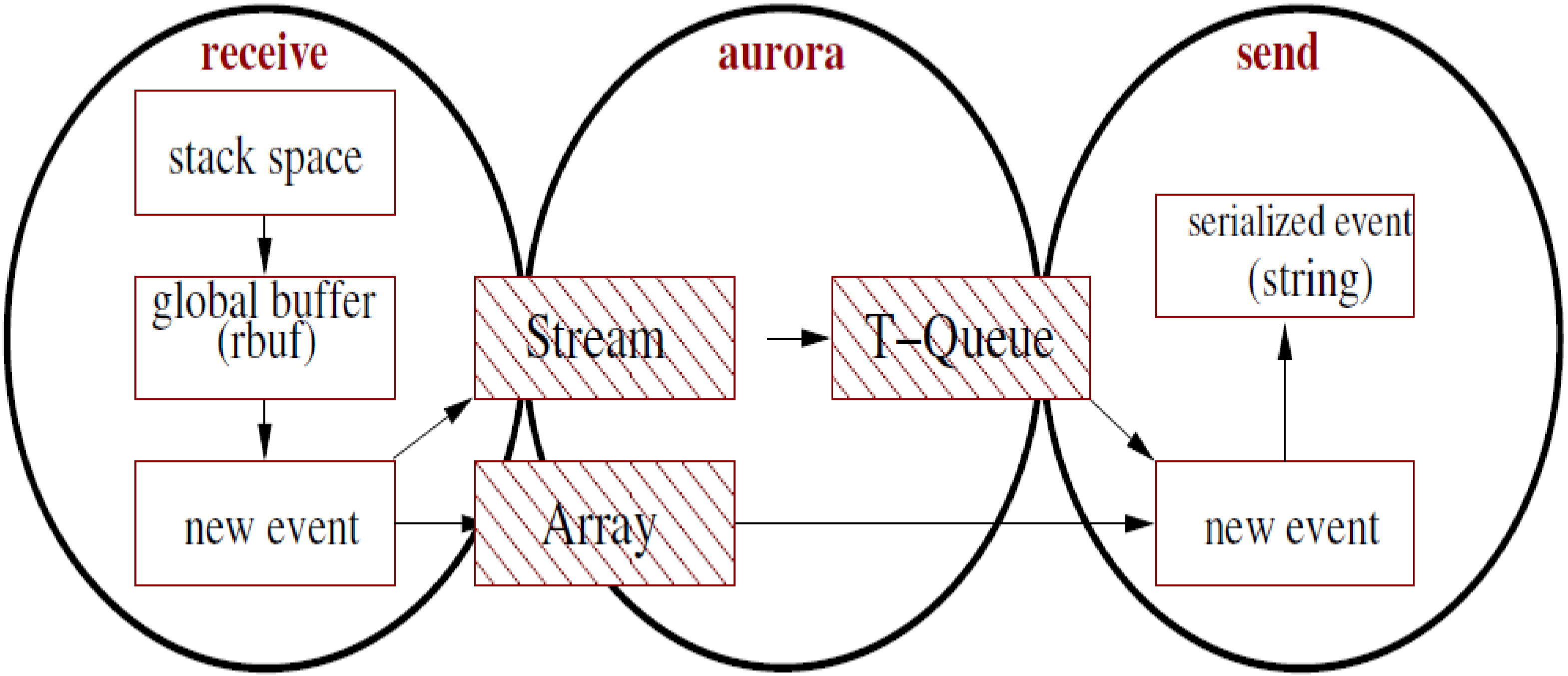
- Message queuing on the send path (1)wbuf (2)D-List

Message Queuing (Asynchronity)



- Message queuing on the send path (1)wbuf (2)D-List
- If network is slow or failure downstream
- Fast networks and reliable hardware
  - Prepare event->send event->prepare next event …

New Stack without Message Queuing



Buffer Management

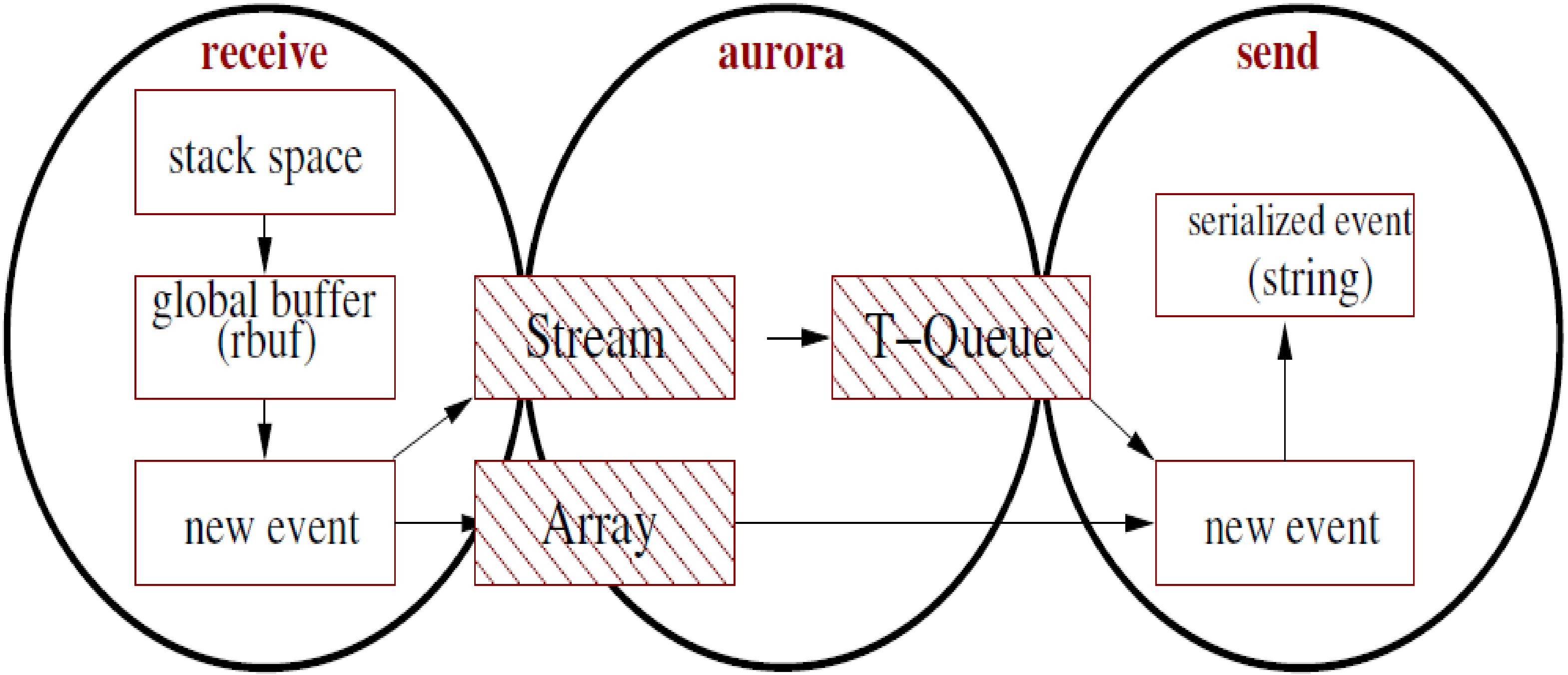
(across threads/modules)

(buffer_ptr, size) tuple

- Pass a pointer to buffer and size
- Need to manage buffer across modules

Copy the Buffer

- Copy data in buffer provided by other module
- Each module does its own buffer



Buffer Management

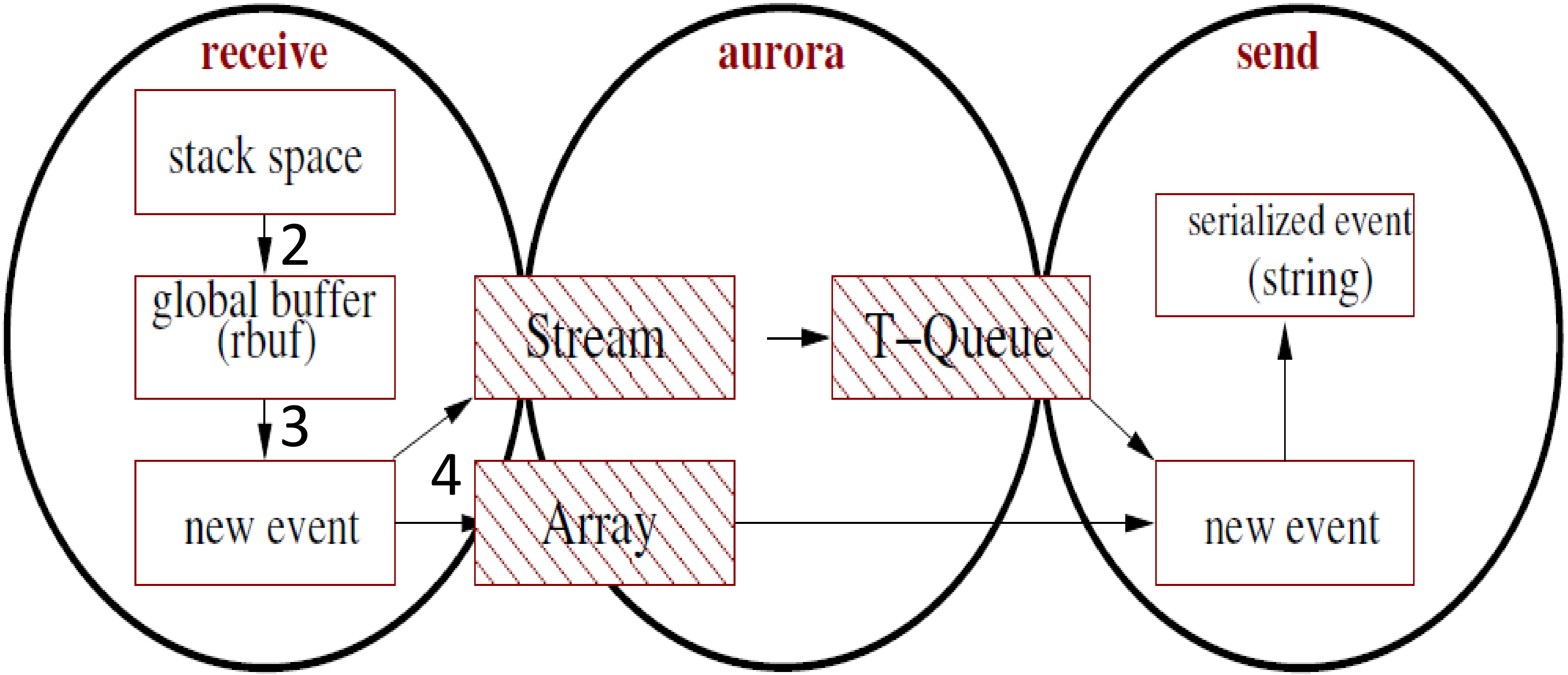
(across threads/modules)

(buffer_ptr, size) tuple

- Pass a pointer to buffer and size
- Need to manage buffer across modules

Copy the Buffer

- Copy data in buffer provided by other module
- Each module does its own buffer



Buffer Management

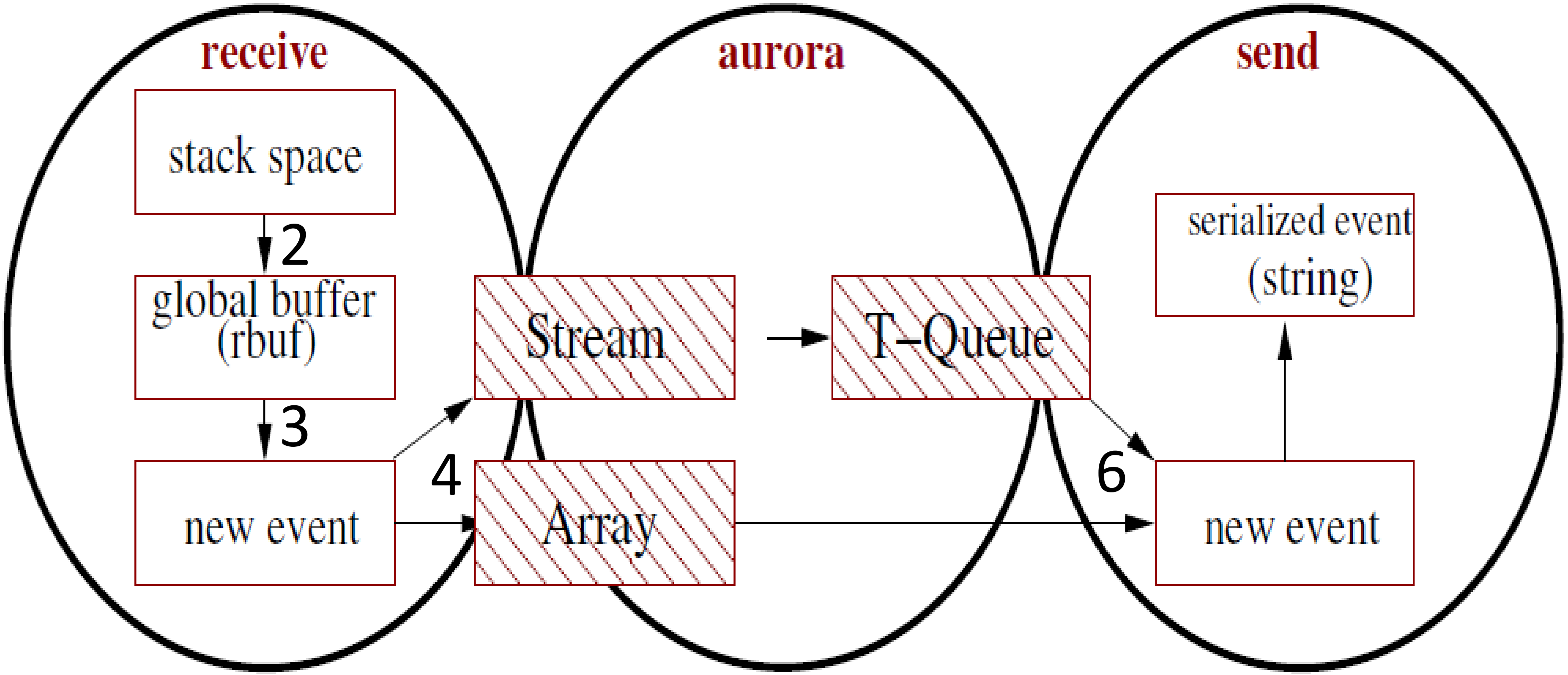
(across threads/modules)

(buffer_ptr, size) tuple

- Pass a pointer to buffer and size
- Need to manage buffer across modules

Copy the Buffer

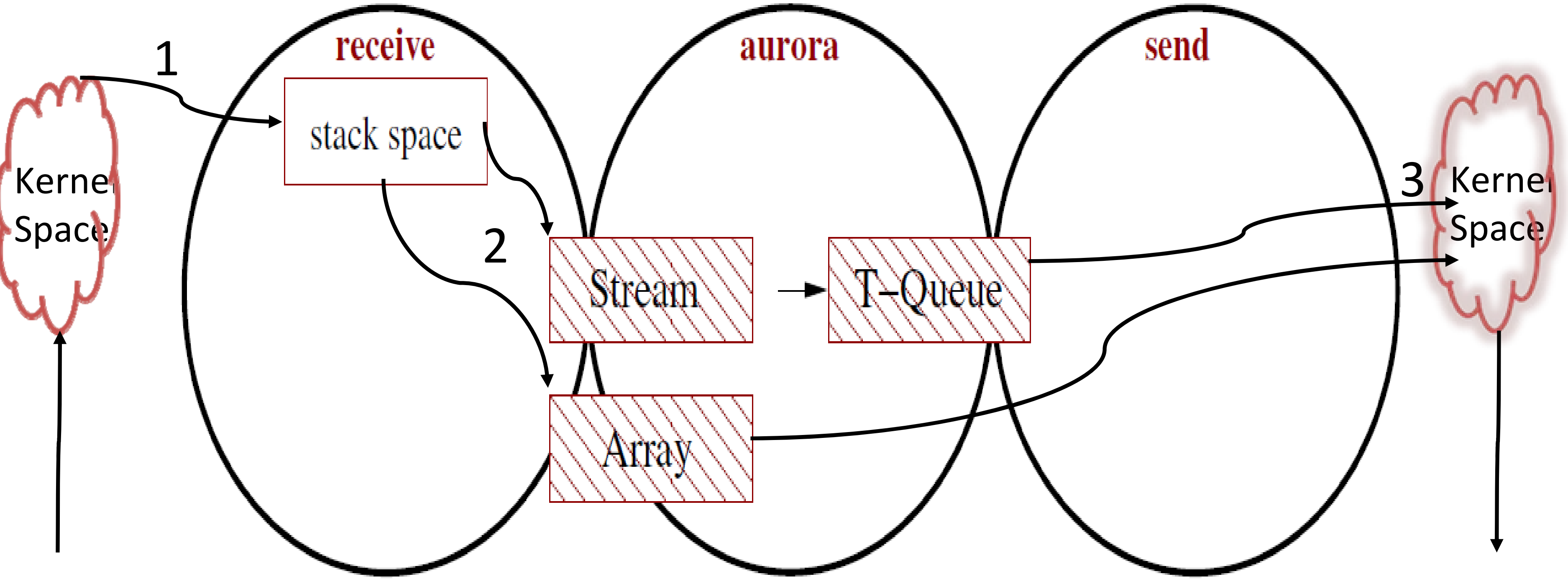
- Copy data in buffer provided by other module
- Each module does its own buffer



Event Serialization

- Serialization
  - Communicate events in binary form (machine independent)
  - Event is scattered in memory
  - Collect the event in a contiguous area in memory
- Alternative?
  - Communicate structure not bytes
  - Structure such as event size, field boundaries
  - Minor increase in network traffic
  - Saves some large memory copies

New Stack without Serialization and with Proper Buffer Management



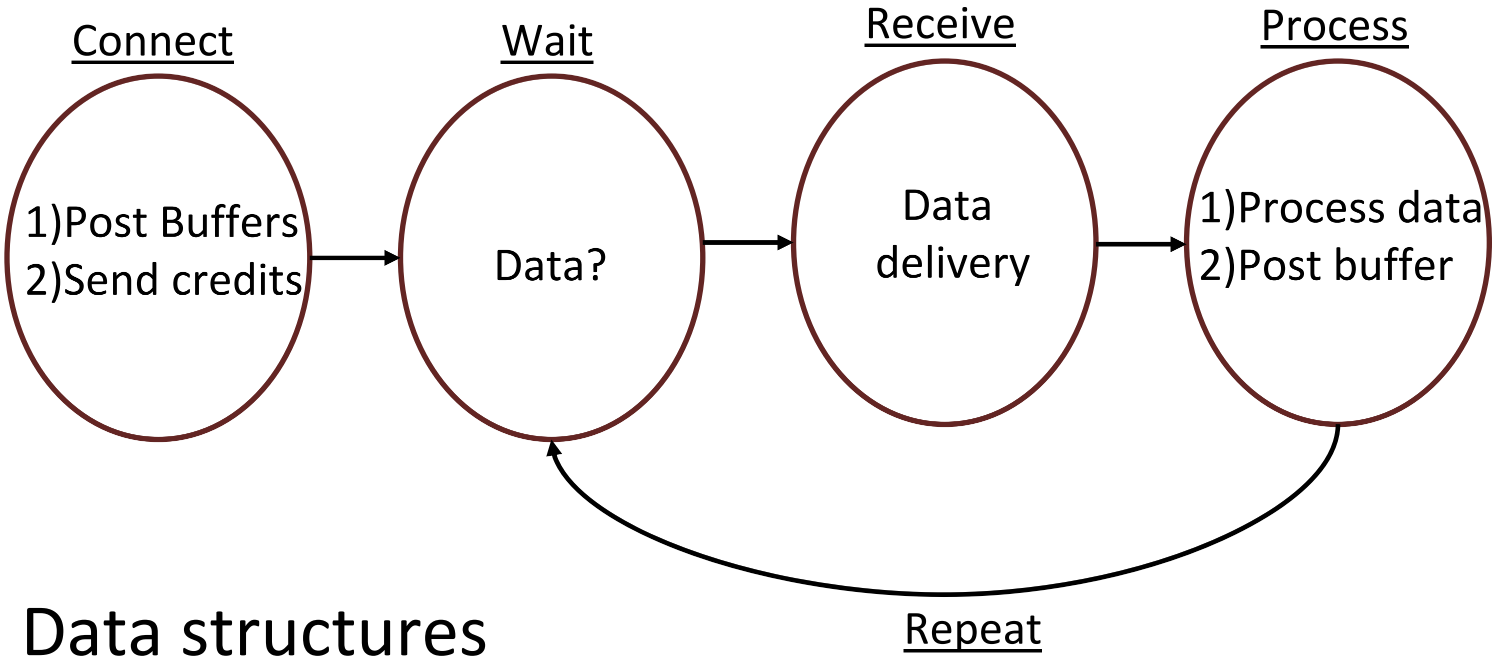
Socket based Network Communication

- Socket based communication uses TCP/IP
- TCP/IP has known overhead
  - Copies in the send and receive path
  - Protocol processing overhead
- User-level network protocols (MX from Myricom)

  +Bypasses the kernel layer and spare CPU cycles

  -Specialized hardware
- How they work?
  - Release (post) buffers (user-space) and inform the sender
  - Sender directly fills the buffer with event data
  - Flow control protocol is custom

Protocol for User-level Communication



- Data structures
  - A circular queue
  - A credit counter (in the process state)

Outline

- ✓ Introduction and Motivation
- ✓ The Borealis Stream Processing Engine
- ✓ Optimizations
- Evaluation Methodology
- Results
  - Throughput
  - Energy
  - Projections for Future
- Conclusions

Goals of Evaluation

- Impact of optimizations
  - *original* (original Borealis)
  - *tcp-opt* (all optimizations except MyrinetMX)
  - *mx-opt* (all optimizations)
- Impact of various parameters
  - Tuple size (128, 1024 and 4096 Bytes)
  - Event size (128 Bytes to 128 KB)
  - Number of instances (1, 4 and 8)

Query Graph

- Two filter operators in a chain
- A source of tuples per instance of Borealis
- A receiver of tuples per instance of Borealis
- Distributed setup of Borealis
- Total of four servers

Experimental Platforms

Setup-A

- One Intel Xeon Quadcore (X3220)
- 8 GB of DRAM
- 10 Gbit Ethernet NIC from Myricom
- Myricom Switch

Setup-B

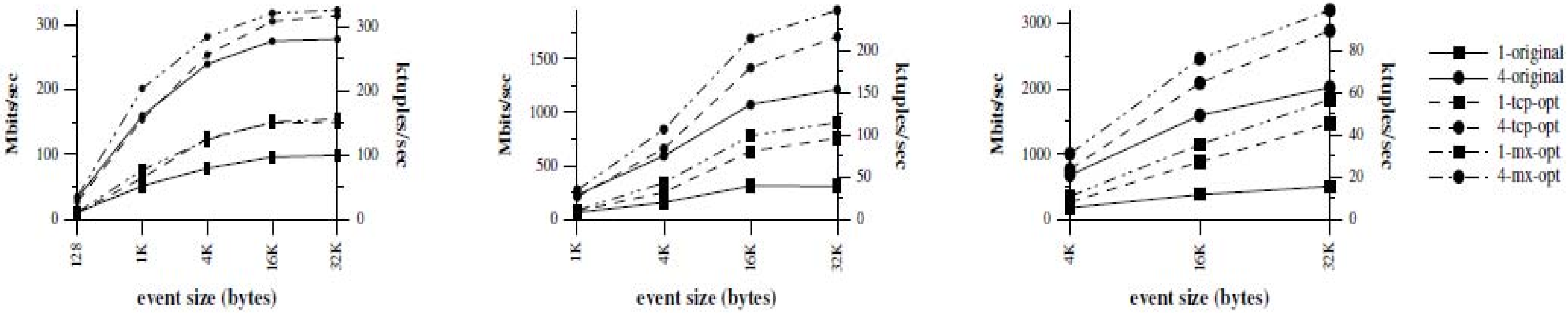
- Two Intel Xeon Quadcore(E5620)
- 12 GB of DRAM
- 10 Gbit Ethernet NIC from Myricom
- 10 Gbit HP ProCurve 3400cl Switch (does not operate with mx-opt)

Estimating Energy Consumption

- Simple model B*u+I
  - I is idle energy and
  - B is busy energy
  - u is CPU utilization
- Use averages for B, I and u (for a large number of events)

Throughput Results

Setup-A

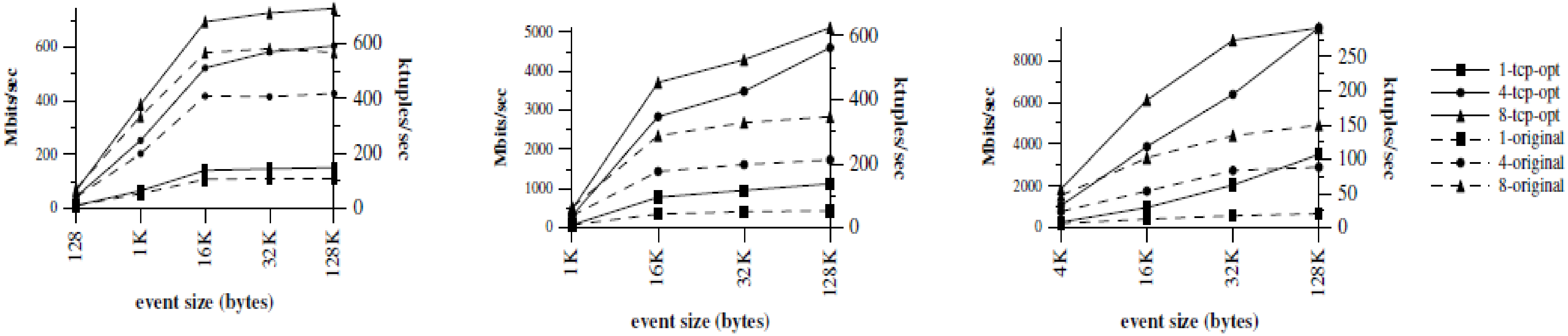


128 Bytes

1 KB

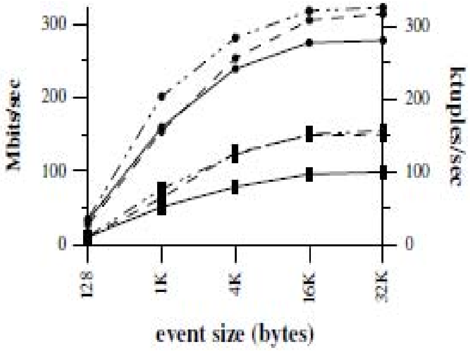
4 KB

Setup-B

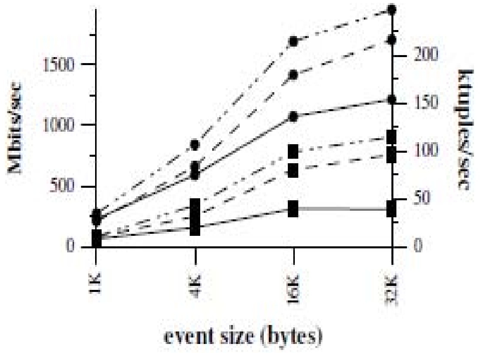


Throughput Results

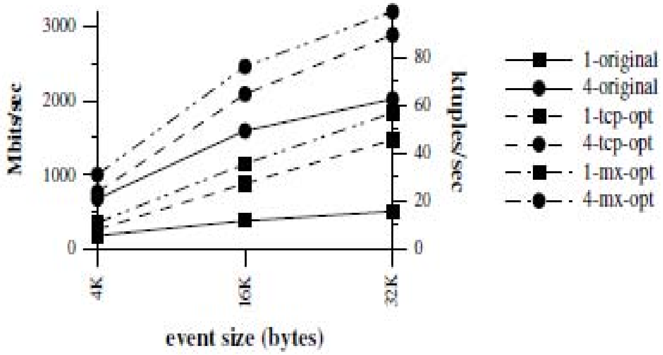
Setup-A



128 Bytes

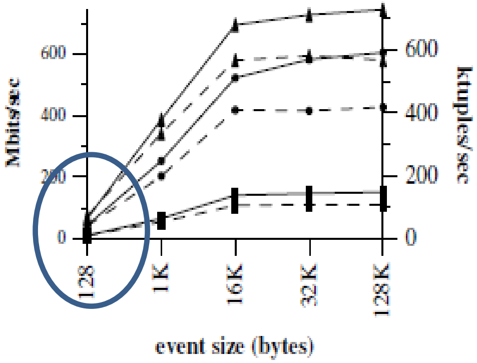


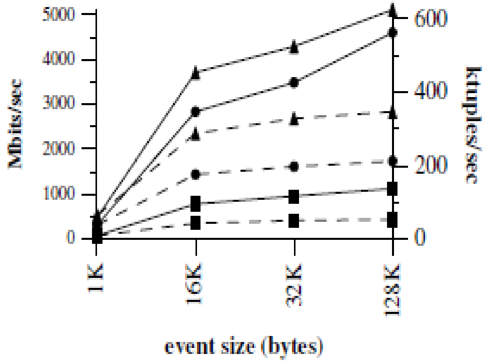
1 KB

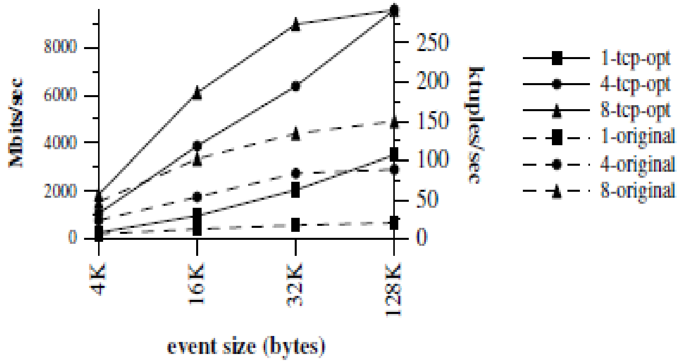


4 KB

Setup-B



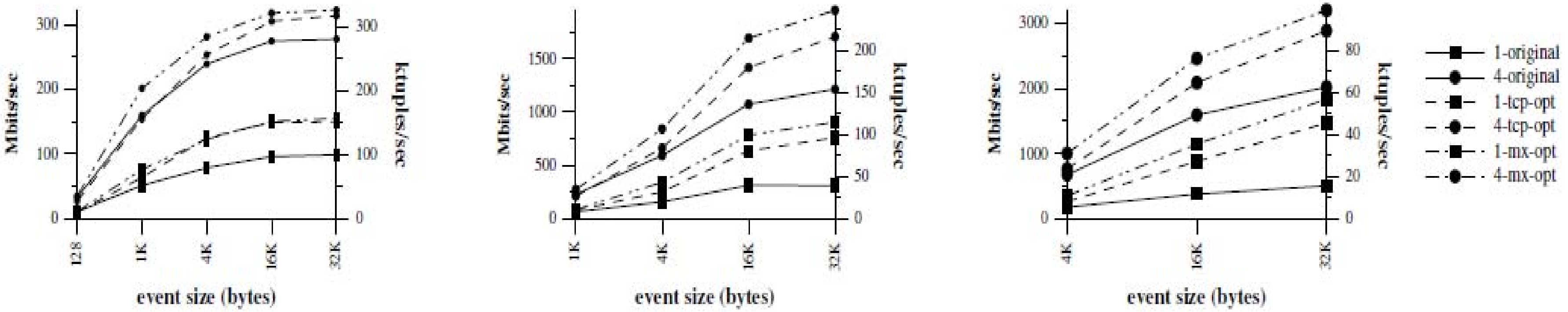




○ No substantial gain

Throughput Results

Setup-A

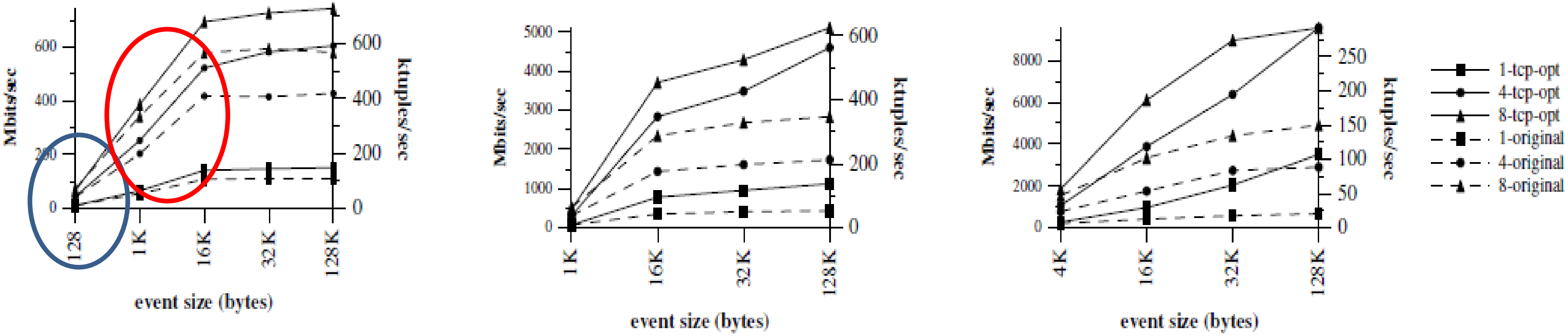


128 Bytes

1 KB

4 KB

Setup-B

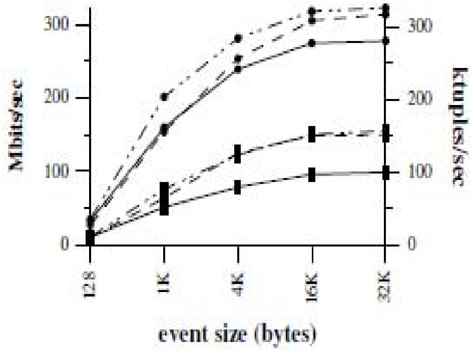


○ No substantial gain

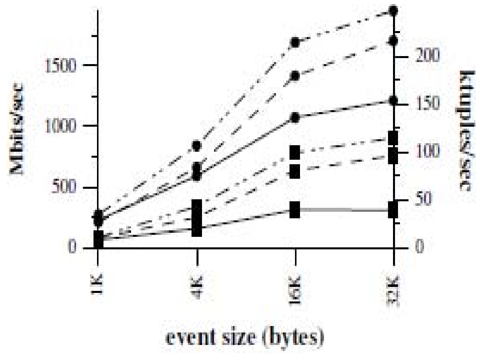
○ High throughput and gain

Throughput Results

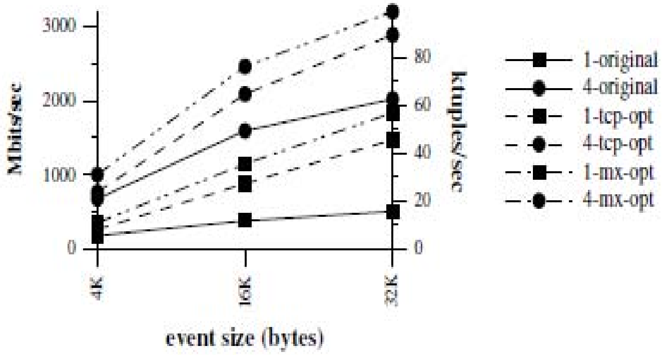
Setup-A



128 Bytes

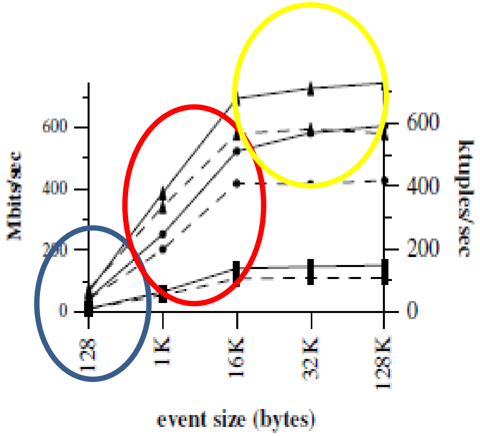


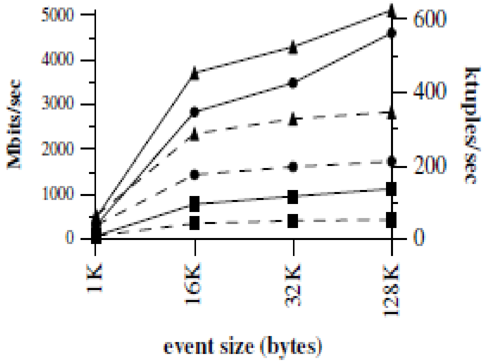
1 KB

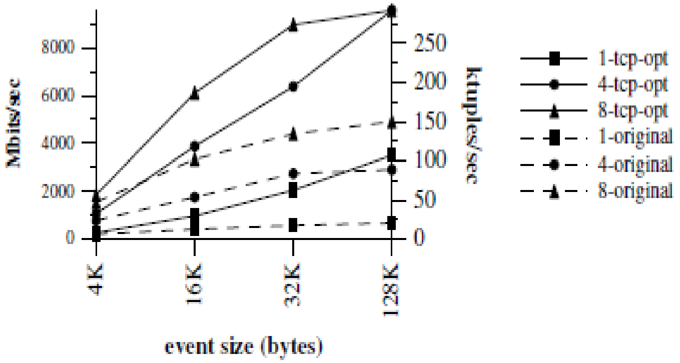


4 KB

Setup-B



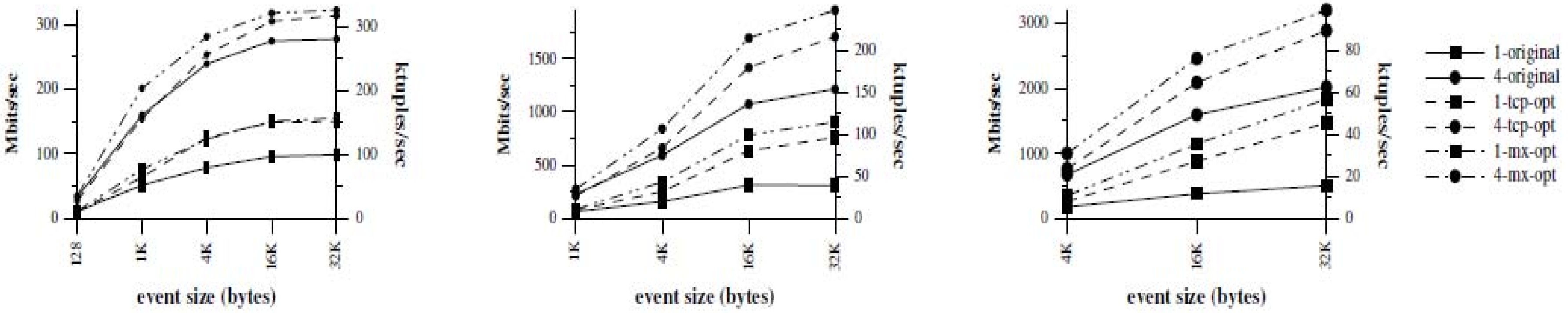




○ No substantial gain ○ High throughput and gain ○ High gain but throughput becomes flat

Throughput Results

Setup-A

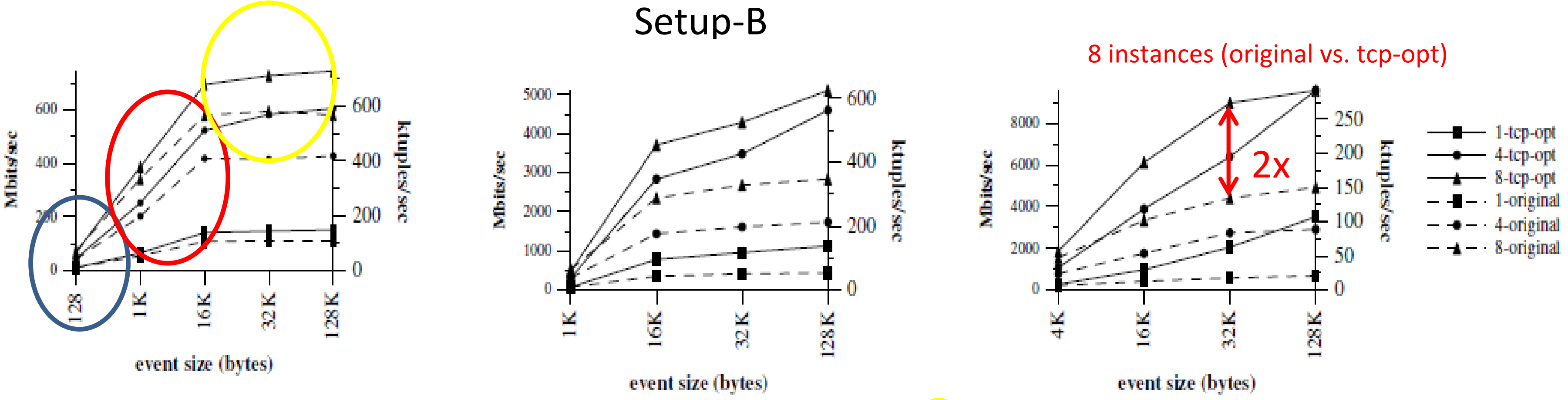


128 Bytes

1 KB

4 KB

Setup-B



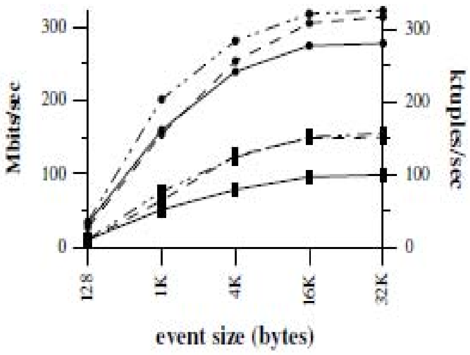
8 instances (original vs. tcp-opt)

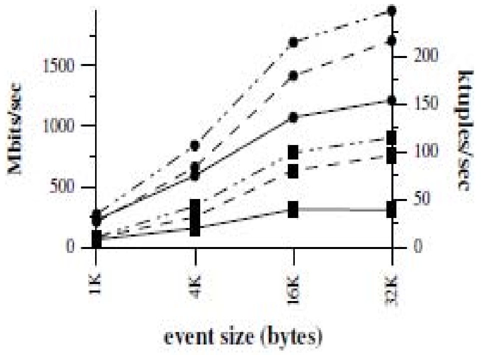
○ No substantial gain ○ High throughput and gain ○ High gain but throughput becomes flat

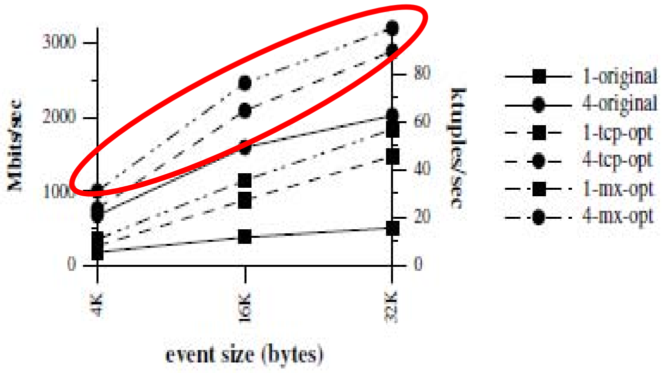
Throughput Results

Setup-A

4 instances (tcp-opt vs. mx-opt)







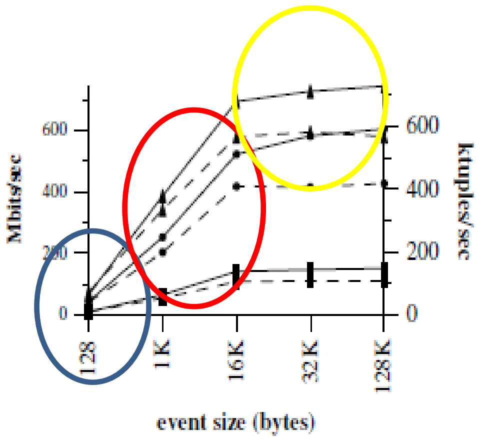
128 Bytes

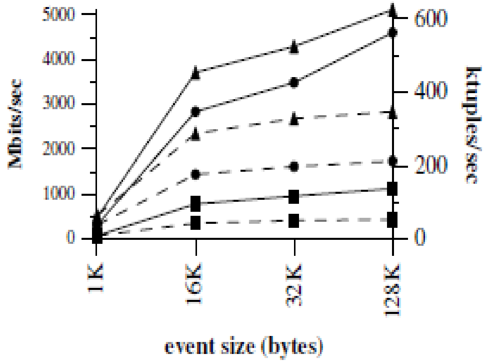
1 KB

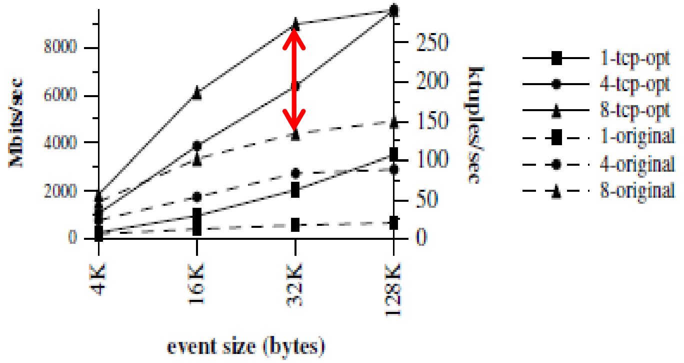
4 KB

Setup-B

8 instances (original vs. tcp-opt)





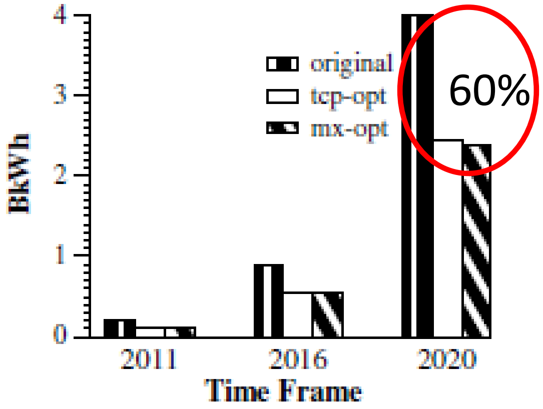


○ No substantial gain ○ High throughput and gain ○ High gain but throughput becomes flat

Summary

- I Million tuple/s (128 Byte tuples)
- 10 Gbits/s (4096 Byte tuples)
- Large events, large tuples, 4 instances (200%)
- Large events, small tuples (50%)
- Small events, small tuples (Negligible)

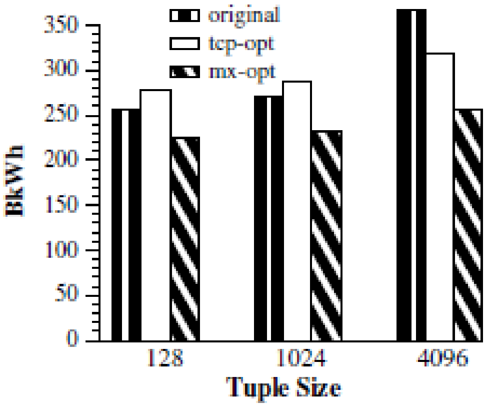
Energy Consumption (Large Events)



4096 Byte Tuples

- Fix amount of data (produced in 2011,2016,2020)
- Fix amount of time to process the data
- Divide data into events (32 KB events)
- 60% reduction with tcp-opt for large tuples
- 3% reduction with mx-opt for large tuples

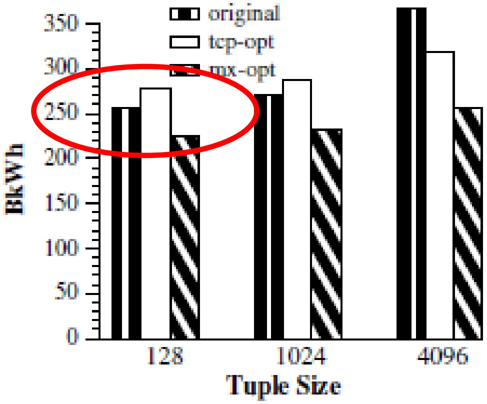
Energy Consumption (Small Events)



2020

Event size=Tuple size

Energy Consumption (Small Events)

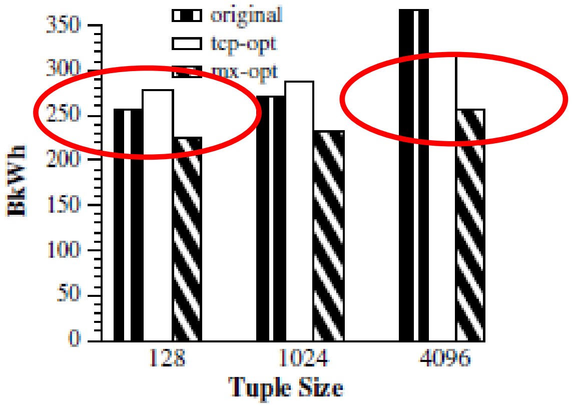


2020

Event size=Tuple size

- Small tuples, tcp-opt has overhead
  - More data communicated per event

Energy Consumption (Small Events)

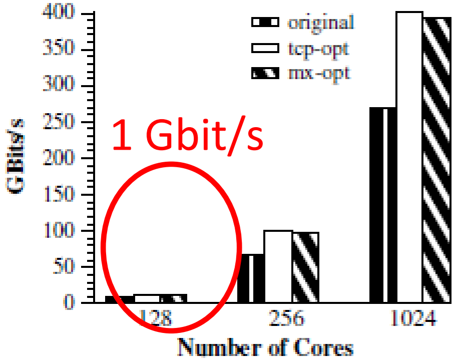


2020

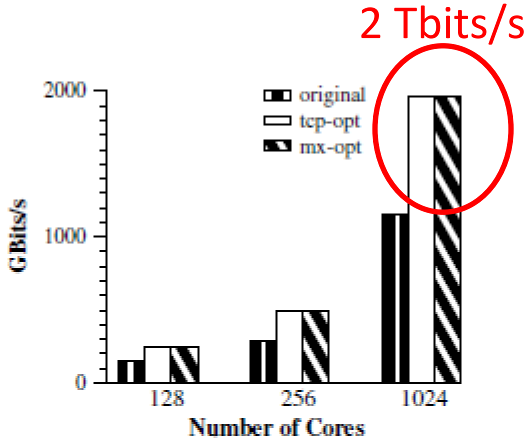
Event size=Tuple size

- Small tuples, tcp-opt has overhead
  - More data communicated per event
- Large events, mx-opt provides 20% reduction

Network Bandwidth Projections



128 Byte Tuples



4096 Byte Tuples

32 KB Events

- Assume number of cores
- Assume frequency of each core
- Assume processing cycles per byte today
- Full CPU utilization
- Could require up to 2 Tbits/s

Conclusions

- Sources of complexity when providing scale?
  - Provide functionality (heterogeneity, portability)
  - Ease of design
  - Support (old) assumptions running on (modern) hardware
- Possible to restructure event-based stacks for scale
  - 1 Million tuples/s (small tuples)
  - 10 Gbits/s (Large tuples)
- Reduction in energy and infrastructure cost
- 2 Tbits/s needed from supporting infrastructure in 2020

Thank you for your attention! Questions?

Shoaib Akram
shbakram@ics.forth.gr
Foundation for Research and Technology – Hellas (FORTH)
Institute of Computer Science (ICS)