Xenrelay: An Efficient Data Transmitting Approach for Tracing Guest Domain

Hai Jin, Wenzhi Cao, Pingpeng Yuan, Xia Xie
Cluster and Grid Computing Lab
Services Computing Technique and System Lab
Huazhong University of Science & Technology
430074, Wuhan, China

Contents

Introduction

- **Widespread virtualization utilization motivates deeper investigation of the performance implications of virtualization.**
- **Trace analysis is an important technology**
- **Designers of trace tool running in virtualization environment has to consider:**
  - Each Domain has its own address space
  - Small sized trace data and frequent trace events
  - trace data be handled in user space

Introduction

- **Xenrelay**
  - transfers large amounts of data from the guest domain kernel to the privileged domain user-space
  - Static shared memory and two mapping methods
  - A non-notification mechanism

Related work

- Several performance tools are implemented in Xen
  - Xenoprof
    - a system-wide statistical profiling toolkit
  - XenMon
    - performance monitoring tool
    - XenMon collects event log with Xentrace
- Xenrelay is based on *relayfs*

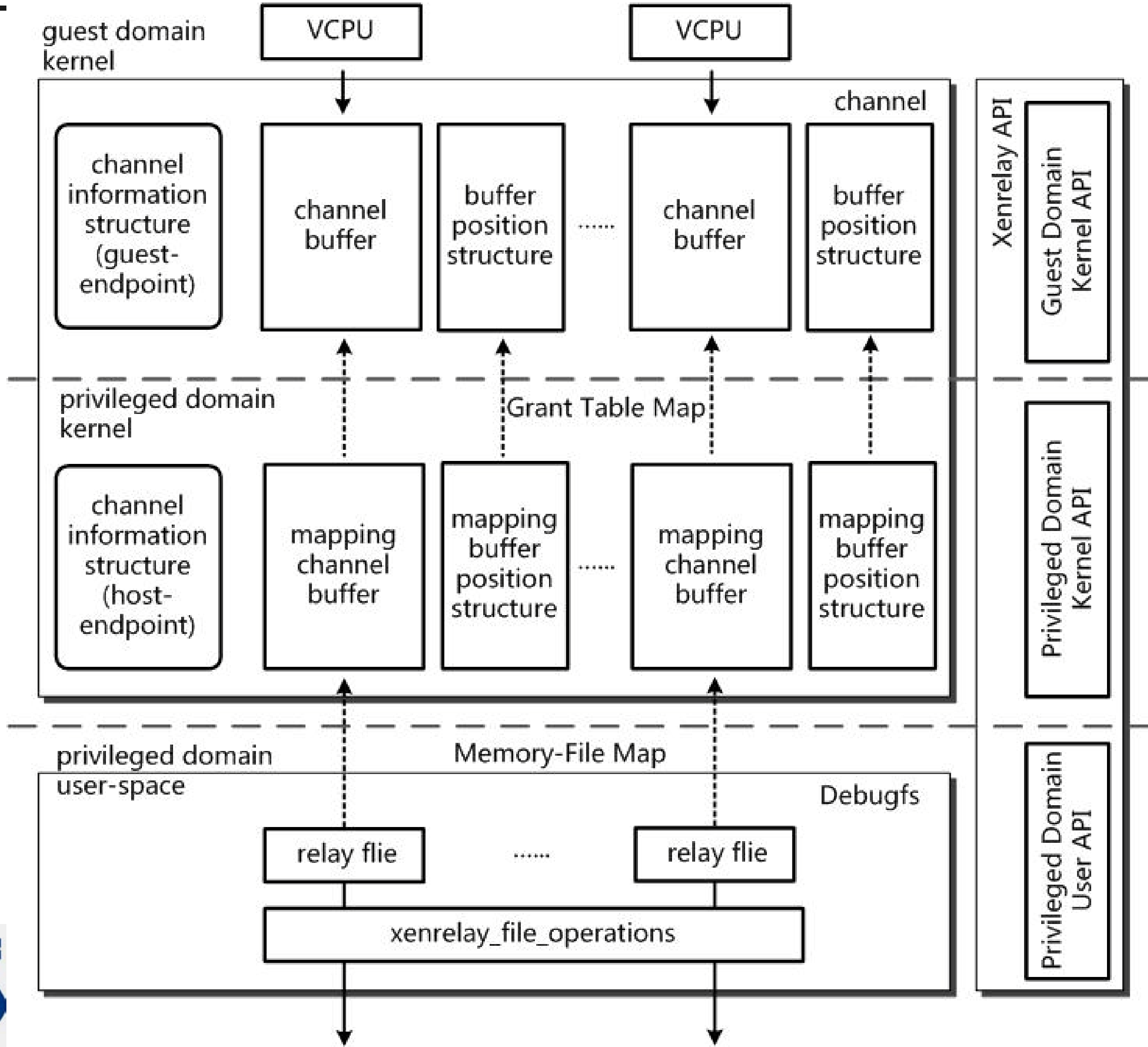
Related work

- Shared memory buffers
  - XenSocket
  - Xway
  - XenLoop
- Those approaches fall short of providing trace data transmission
  - More hypercall, more overheads
  - Head and other data for packing data
  - trace tool should not rely on subsystem

Xenrelay Overview

- **Main issue of Xenrelay**
  - create a channel through three layers
  - control the data transmission without lock and notification
- **Xenrelay contains:**
  - relay channel
  - API

Xenrelay Overview



Design and implementation

- Transmitting at the high speed
- Channel design
  - channel buffers, buffer position structure, channel information structure
  - channel buffer is a producer-consumer circular buffer
  - blocks until the transaction is complete
  - multiplexed I/O and Poll()

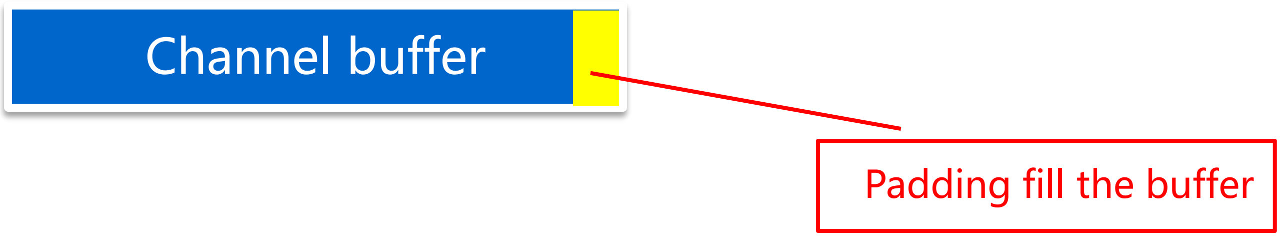
Design and implementation

- **Channel implementation**
  - Allocate high-order address
  - Map no-consisted pages into consisted pages
  - aggregate *buffer position structure* into a memory page
  - assigns an explicit size to data items
- **Channel bootstrap and teardown**
  - XenStore mechanism

Design and implementation

Data relay

- Does not split data
- Does not parse the data



Discussion

- Operation sequence
  - Open -> Connect -> Write -> Read -> Disconnect -> Close
  - Open -> Write -> Connect -> Read
    - be safe
    - easy to make data overflow
  - Close -> Disconnect
    - memory leak in guest domain

Discussion

- Data overflow
  - write faster than read
  - suspends writing and cause loss of new data
- Relay file
  - only the data in the end of file is valid

Experimental testbed

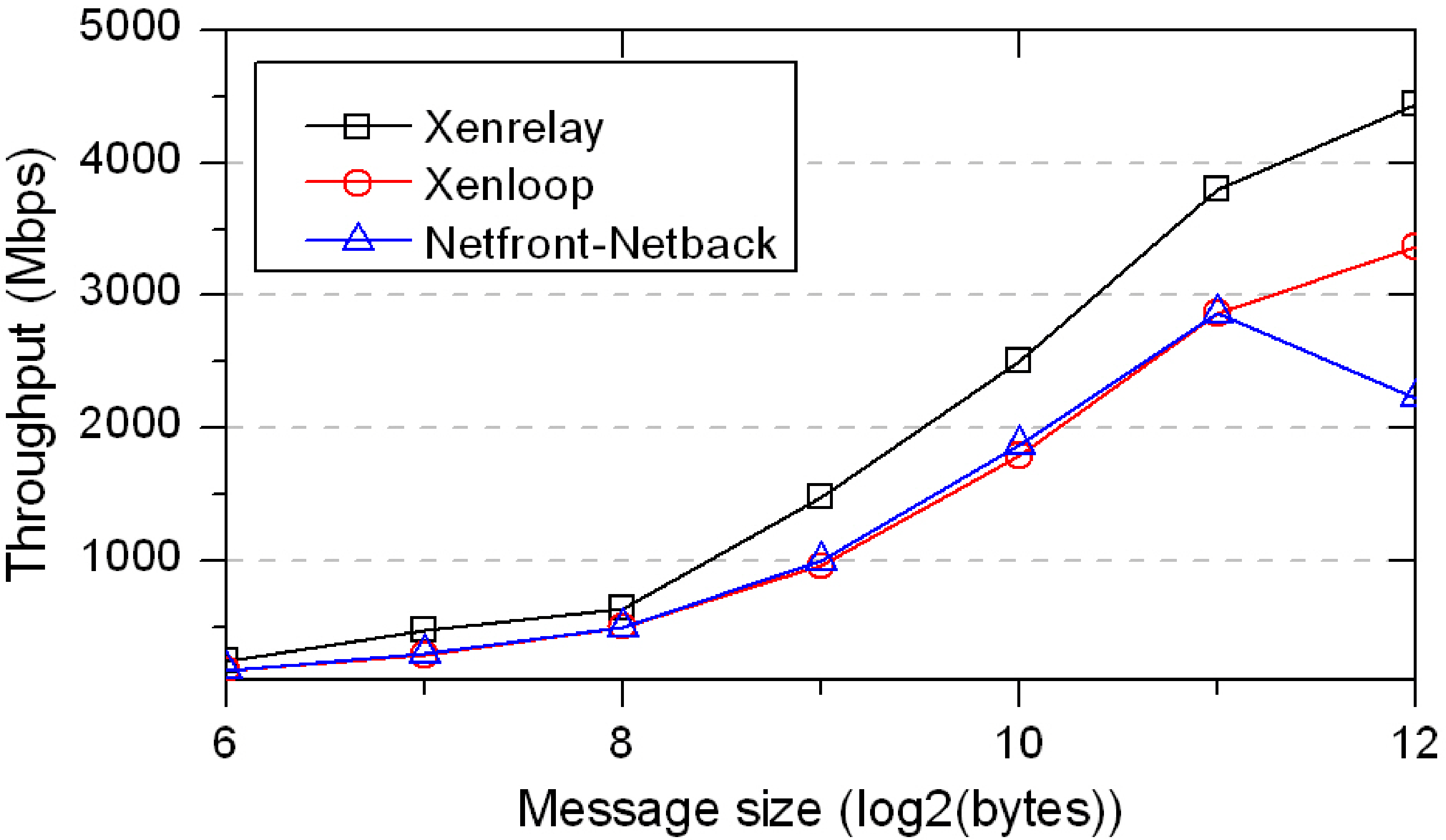
- Experiment setup
  - A server-class machine
    - two Intel Xeon E5310 CPUs (1.6 GHz and four cores CPUs)
    - 4GB Memory
    - 500GB SATA disk (rotation speed is 7200rps)
  - Xen version 3.3.1 and linux kernel version 2.6.18_X64
  - Each guest domain has 1 VCPU, 512MB memory and 10GB disk image

Experimental testbed

- Data transmission scenarios
  - Netfront-Netback
  - XenLoop
    - ✓ netperf's TCP_STREAM test
  - Xenrelay
    - ✓ Move a fixed size of data in relay channel from DomU to Dom0 repeatedly
    - ✓ 100MB data in total

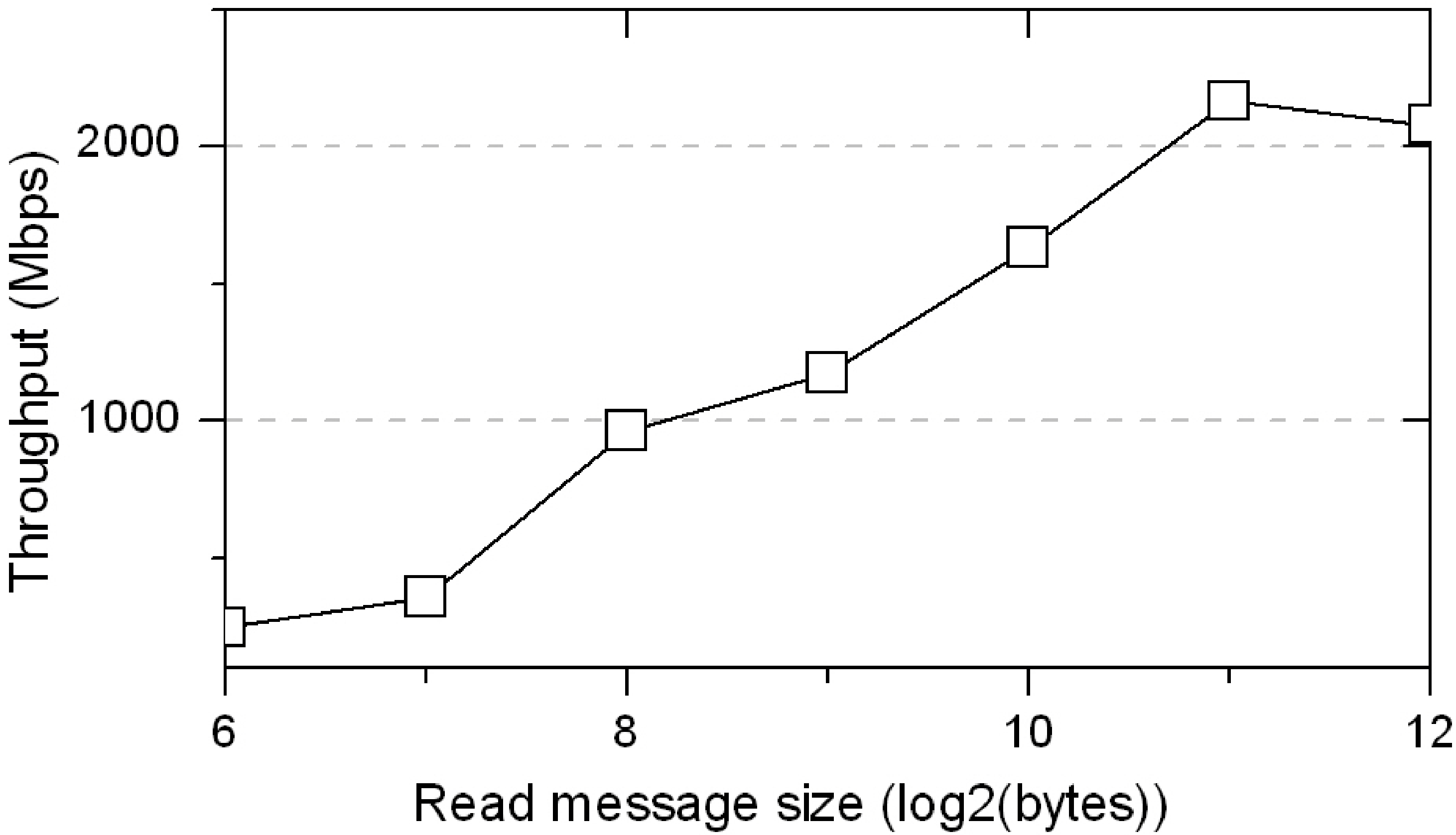
Experimental results

- **Throughput VS. message size**
  - In Xenrelay, reading message size = writing message size



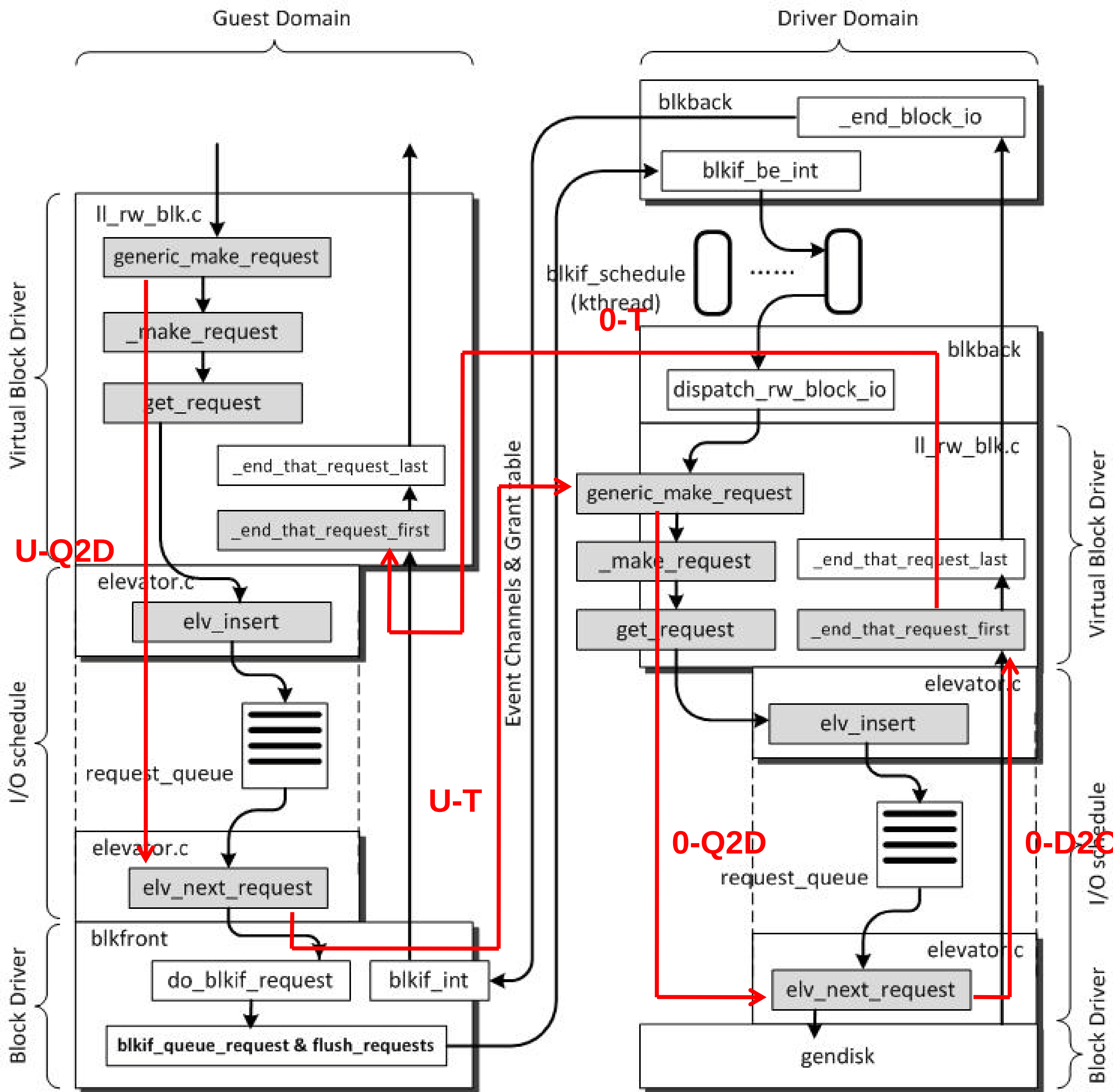
Experimental results

- **Throughput VS. reading message size**
  - writing message size = 2^6 bytes
  - reading message size >= writing message size



Applications of Xenrelay (1)

- **Use Xenrelay to create a block trace toolkit**
  - In guest domain kernel, traces I/O events and uses Xenrelay to transfer it
  - In privileged domain kernel, traces I/O events and uses relayfs to transfer it
- **Trace point**
- **Trace data**
  - device number, the event time stamp, the start sector number, the number of handle block, the event type and the operation type



U-Q2D: *generic_make_request to elv_next_request*

U-T: *elv_next_request to generic_make_request in dom0*

0-Q2D: *generic_make_request in dom0 to elv_next_request*

0-D2C: *block device handling request*

0-T: *blkback returning blkfront*

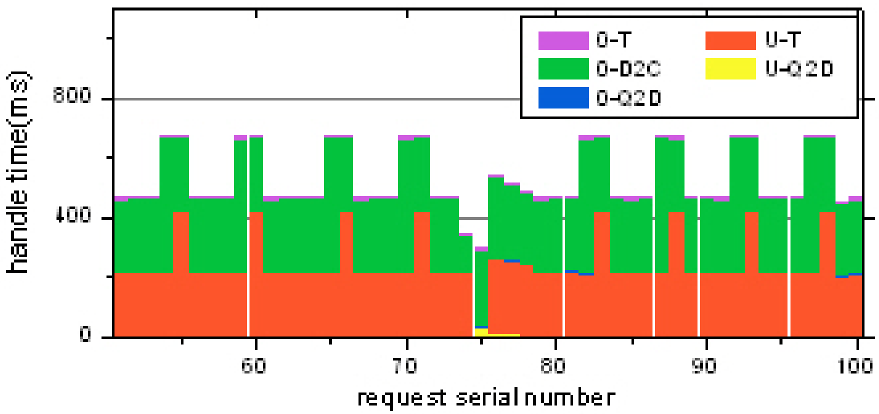
Applications of Xenrelay (2)

- evaluate typical combinations of the I/O schedulers
  - Guest Domain Schedule Test
    - ✓ Change guest domain's Schedule
    - ✓ Fix driver domain's Schedule and use Noop schedule
  - Driver Domain Schedule Test
    - ✓ Change driver domain's Schedule
    - ✓ Fix guest domain's Schedule and use Noop schedule
- A single 100MB file was read

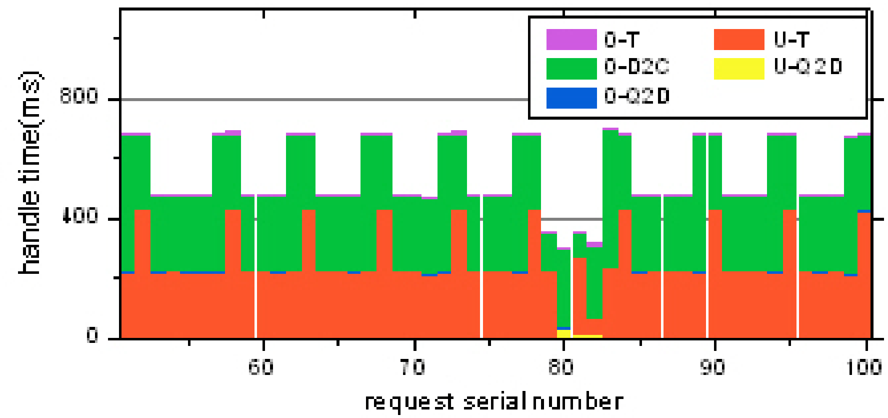
Applications of Xenrelay (3)

Result of tracing

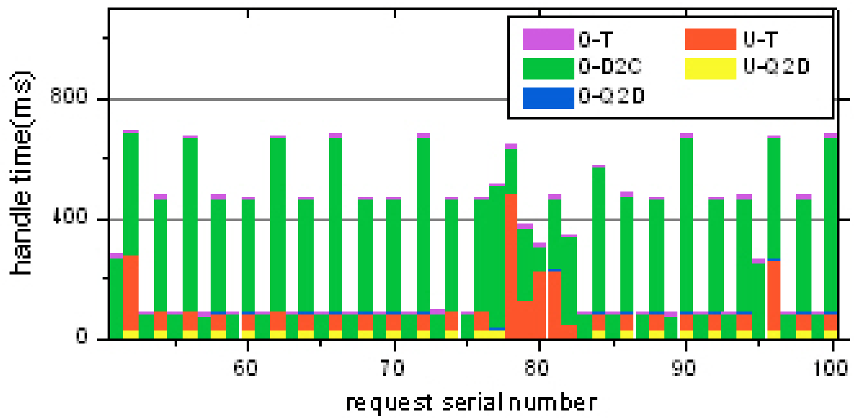
- Guest domain schedule test



Noop schedule



AS schedule



CFQ schedule

***U-Q2D**: generic_make_request to elv_next_request*

***U-T**: elv_next_request to generic_make_request in dom0*

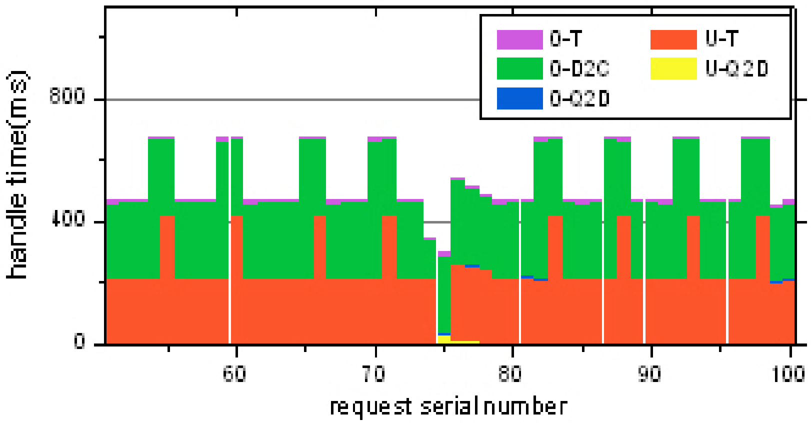
***0-Q2D**: generic_make_request in dom0 to elv_next_request*

***0-D2C**: block device handling request*

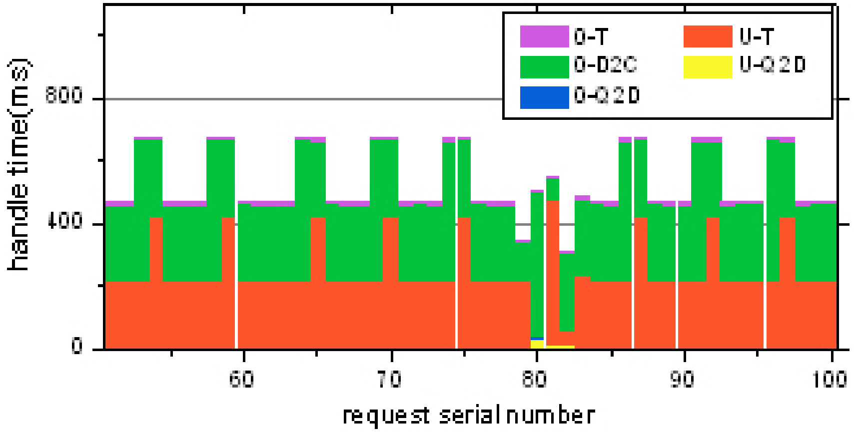
***0-T**: blkback returning blkfront*

Applications of Xenrelay (4)

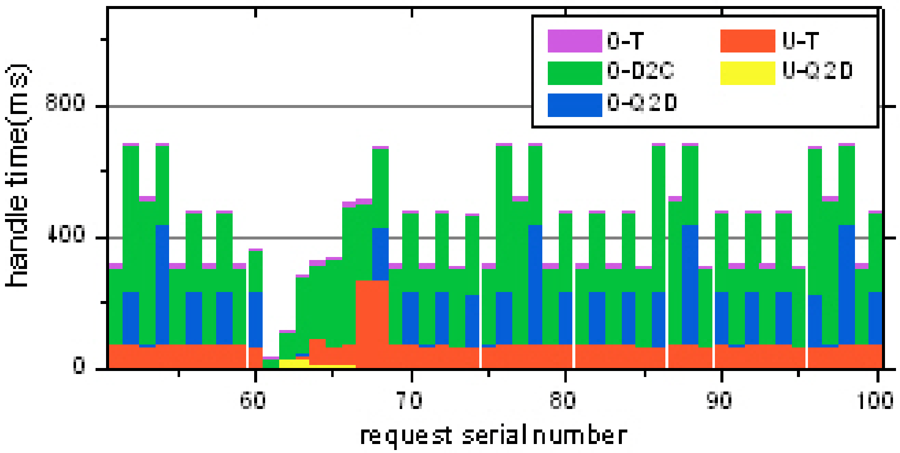
- **Result of tracing**
  - Driver domain schedule test



Noop schedule



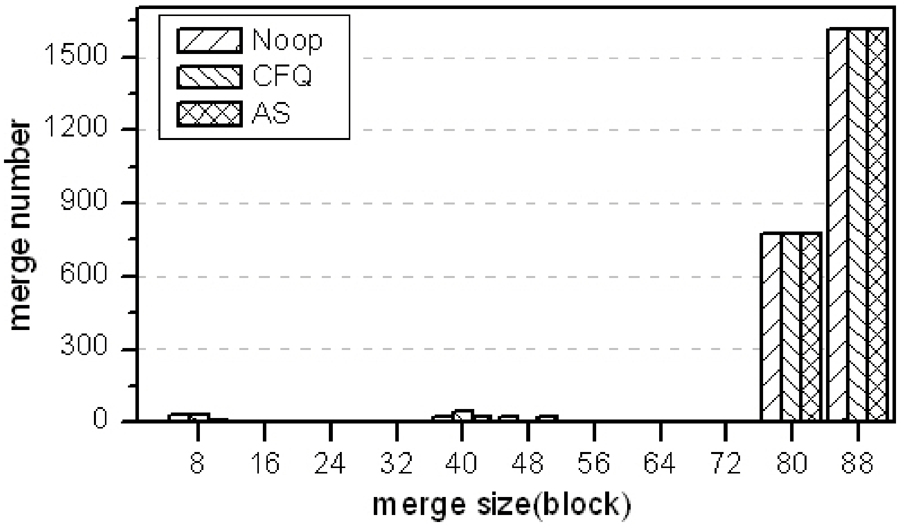
AS schedule

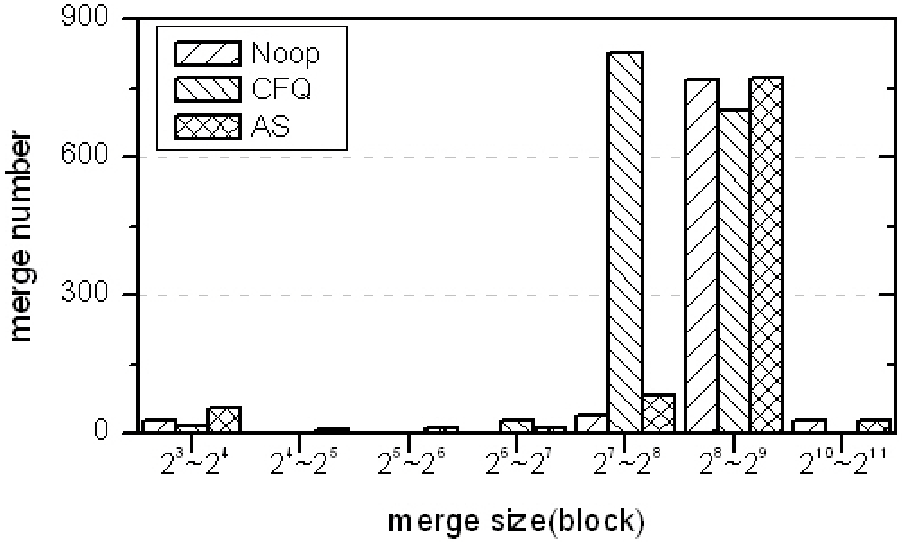


CFQ schedule

Applications of Xenrelay (5)

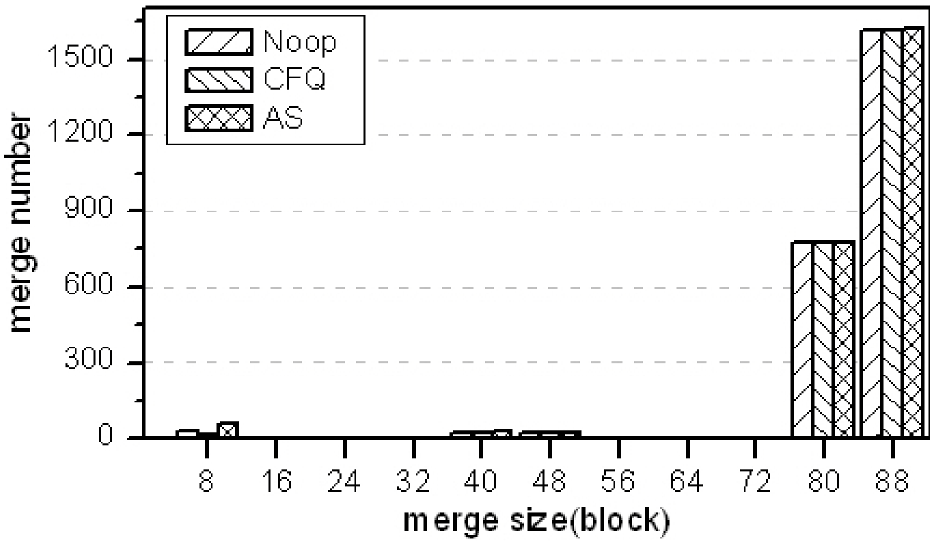
- Merge block number and size
  - Guest domain schedule test

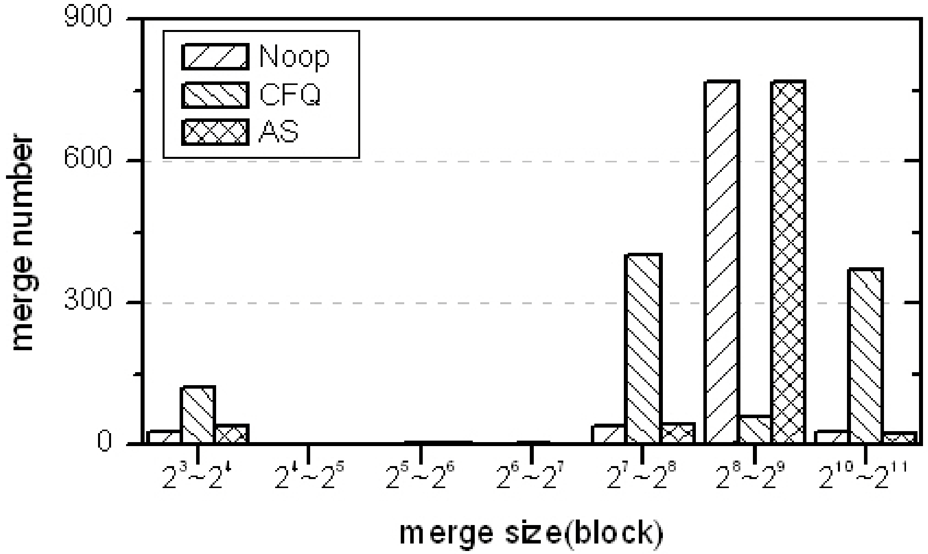




Applications of Xenrelay (6)

- Merge block number and size
  - Driver domain schedule test





Conclusion

- Xenrelay is a unified, efficient, and simple mechanism for transferring data
- Xenrelay provides supports for users who trace subsystems to record and relay data

Thank you!

Questions ?