vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities

June 18, 2009

Byung Chul Tak★♪
Chunqiang Tang★★♪
Chun Zhang★★♪
Sriram Govindan★♪
Bhuvan Urgaonkar★♪
Rong N. Chang★★♪

PENNSTATE 1855

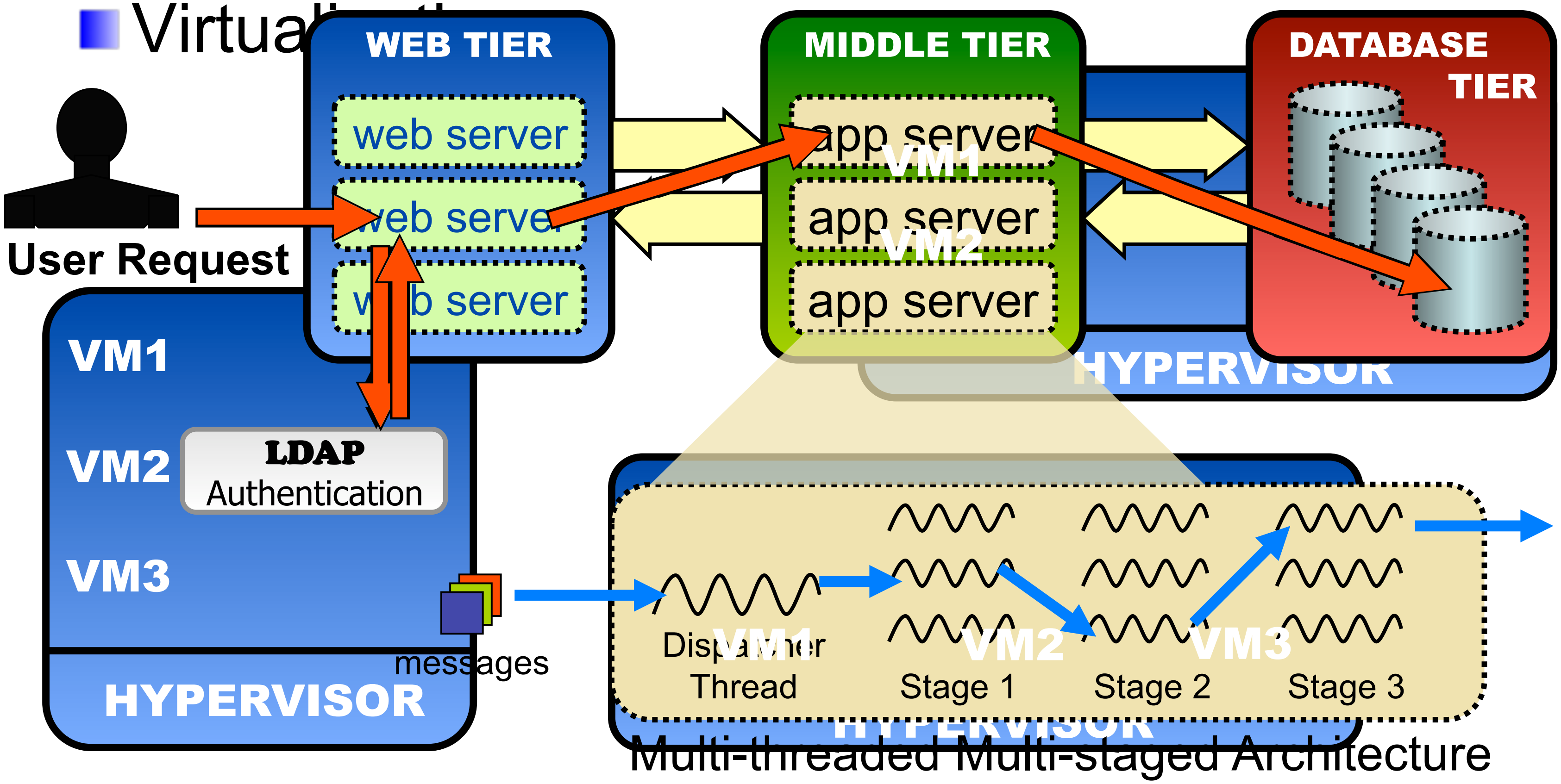
★♪ Pennsylvania State University

IBM

★★♪ IBM T.J. Watson Research Center

Request-Processing Path Discovery

- Enterprise server architecture
  - Three-tiered architecture
    - Increasing complexity from heterogeneity
  - Virtua



Motivation

- *Request-Processing Path* information is critical to managing distributed applications
  - Debugging, analysis, auditing, billing …
- Challenges in obtaining and exploiting the information
  - Develop application-specific middleware
  - Understand logs generated by the middleware
  - Pinpoint the root cause of the problem♪

Existing Solutions

- Statistical inference

General, but NOT accurate, especially for individual Request-processing paths

- Instrumentation-based approach (E.g., Tivoli)

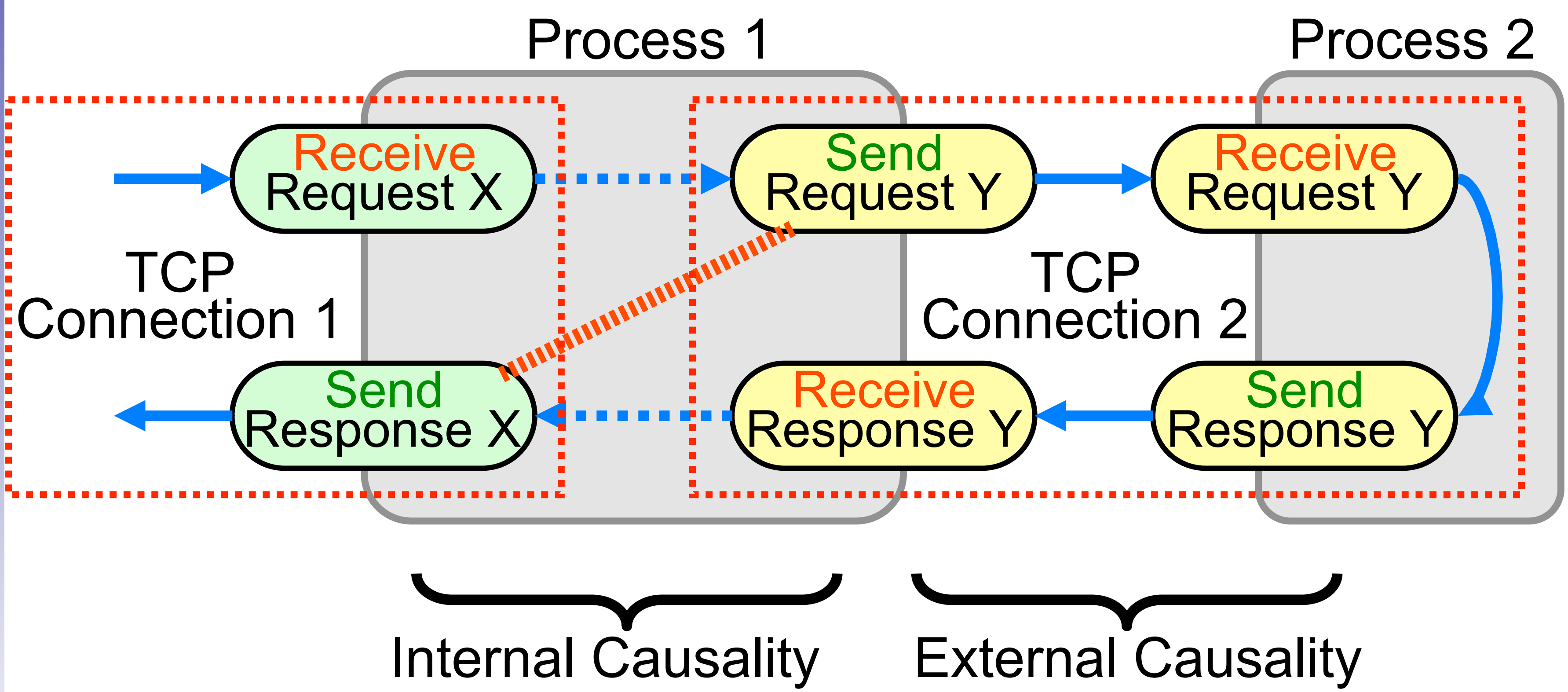
Requires app/MW/OS code changes

vPath Technique

- **vPath discovers:**
  - **Precise** end-to-end request-processing path in a virtualized environment
  - **Without instrumentation** on middleware or applications♪
- Contributions
  - New approach to the path discovery problem
    - Leverage common programming patterns in thread and communication
  - Prototype implementation of the concepts
  - Demonstration of accuracy and completeness

Key Concept of vPath

- Causal Relationships
  - Two types of causality



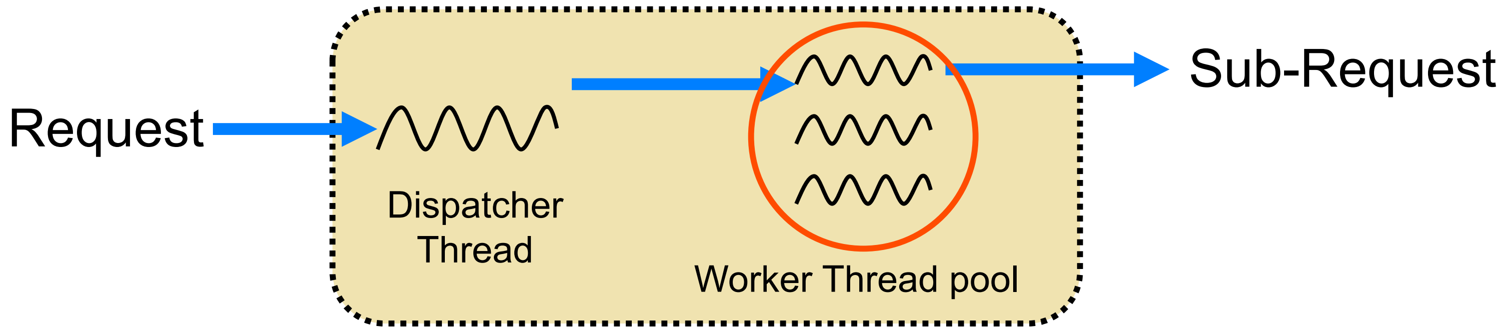
Source of Difficulty

- Message flow
  - Message is used up at arrival
  - Totally new message is assembled
    - Two messages share no common ID
  - Known options
    - Guess → statistical inference
    - Insert ID → instrumentation
- Can we correlate incoming and outgoing me ssages?
  - We consider the execution model♪

Execution Models

Application Model

Thread pattern in Multi-threaded Model



- Single thread is dedicated to the request until final res ponse is sent out

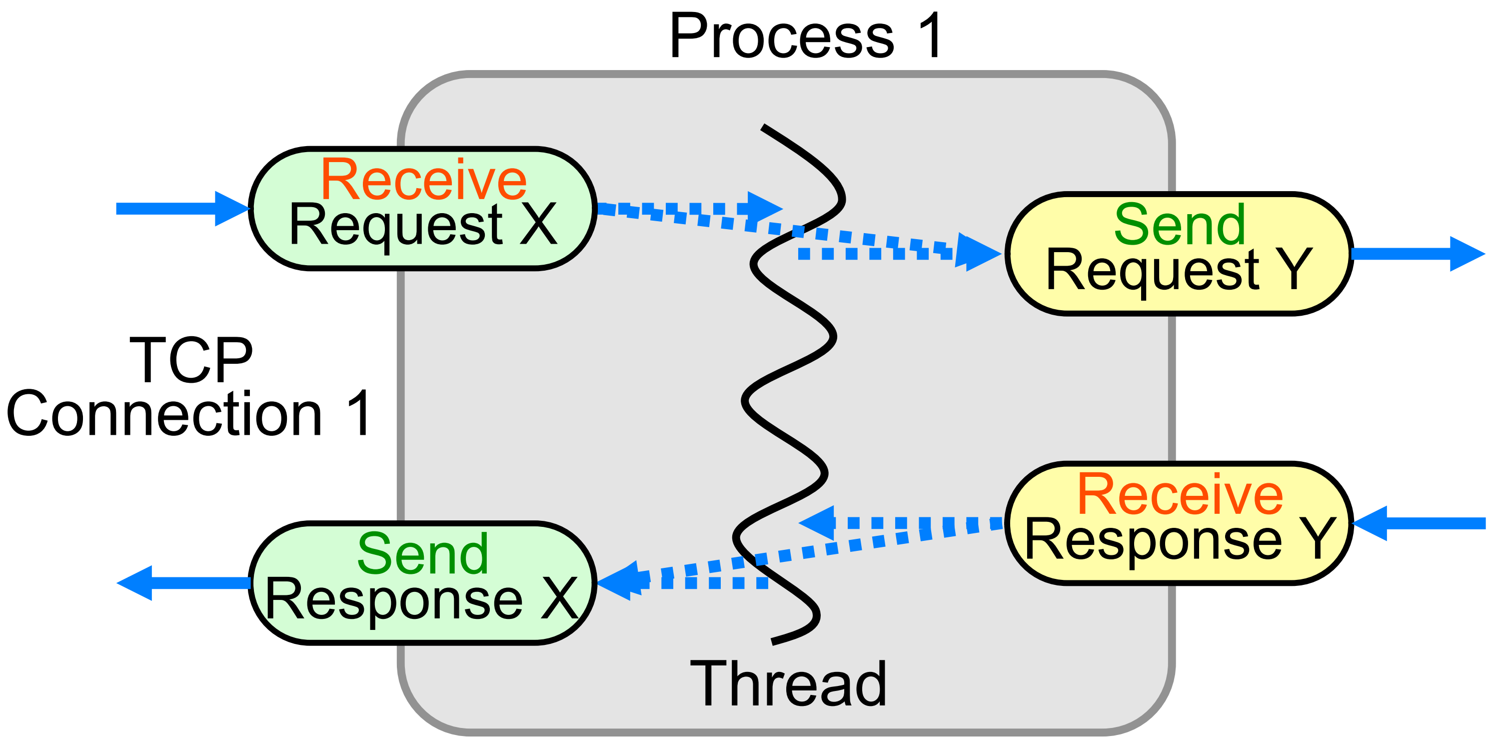
Communication Model

Synchronous communication

- One thread sends a messages and ***blocks*** until it rec eives the reply♪

Identifying Internal Causality

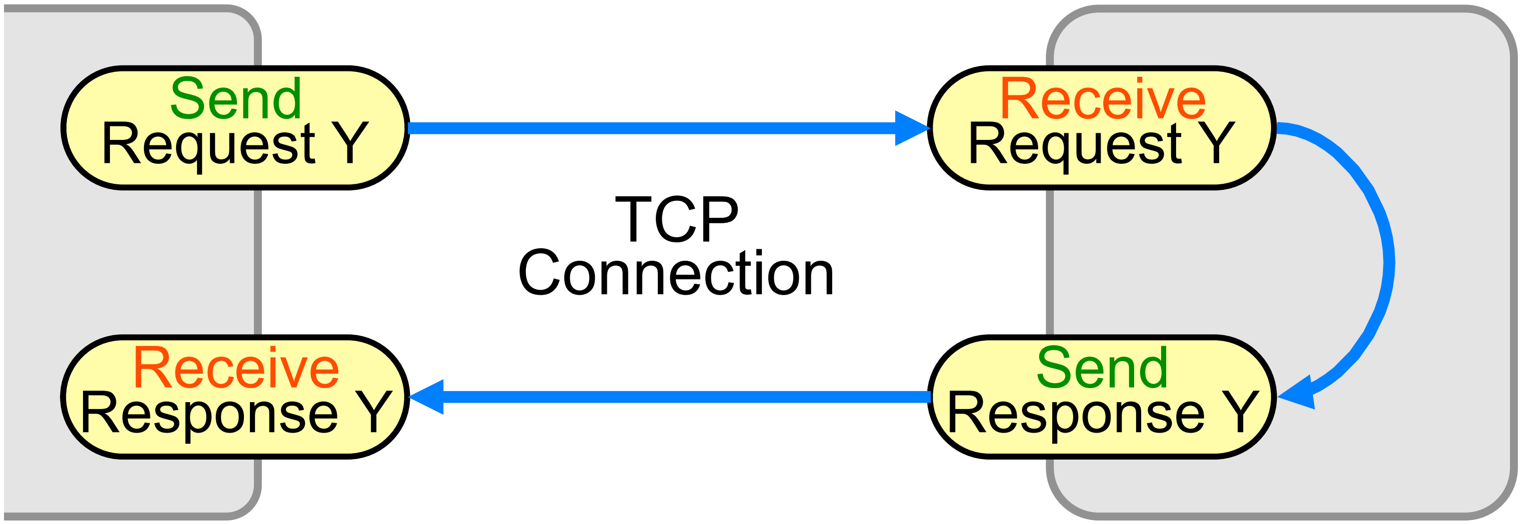
- Causality is carried on to the thread
  - We identify thread from VMM



Identify the thread from the Virtual Machine Monitor

Identifying External Causality

- We use TCP socket information
  - (Source IP, port, Destination IP, port) is compare d and connected

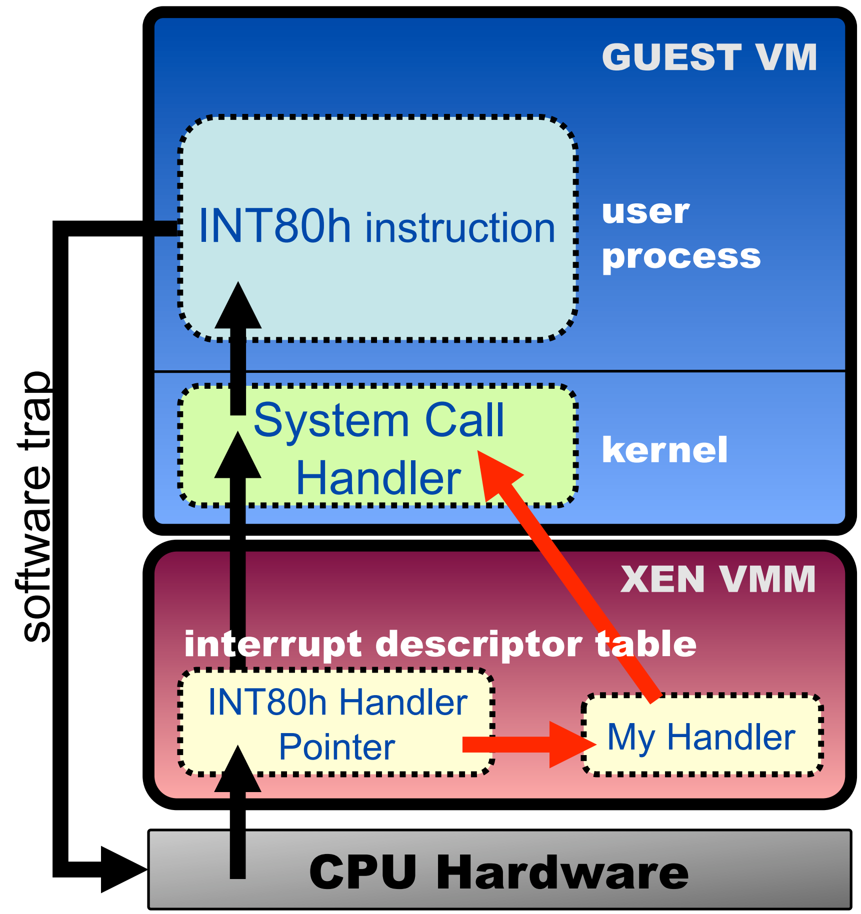


ex: [130.203.8.23:38294, 130.203.8.23:3314]

- We read socket information on Receive and Send eve nts

Implementation

- System Call Interception
  - Intercept system calls by modifying Xen VMM
  - For each system call, get thread identifier
    - EBP register value (stack address)



vPath Implementation 2/2

- Socket info extraction
  - socket – (source IP, port, destination IP, port)
    - This uniquely identifies TCP connection
    - This enables us to correlate events across components
  - Custom hypercall
    - On every target system call, this hypercall is invoked
    - It delivers socket information from Guest Kernel to Xen VMM

Path Data Processing

Log Format

From System Call Interception

Event #	Domain #	Time Stamp	CR3	EBP	EAX	EBX

From Hypercall

Event #	OP Type (R/S)	Domain #	Socket Descriptor #	Local IP Addr & Port	Remote IP Addr & Port

Example

```
0733  Dom1      002780     cr3:04254000     ebp:bfe37034     eax:3      ebx:12
0734  R   Dom1   sd:12   L:130.203.8.24:41845   R:130.203.8.25:8009
0735  Dom1      002781     cr3:04254000     ebp:bfe34b34     eax:146   ebx:11
0736  S   Dom1   sd:11   L:130.203.8.24:80   R:130.203.65.112:2395
0737  Dom2      002780     cr3:04254000     ebp:bff2203f     eax:3      ebx:12
0738  R   Dom2   sd:12   L:130.203.8.24:41811   R:130.203.8.25:8009
0739  Dom1      002781     cr3:04254000     ebp:bfe34b34     eax:146   ebx:11
0740  S   Dom1   sd:11   L:130.203.8.24:80   R:130.203.65.113:3411
```

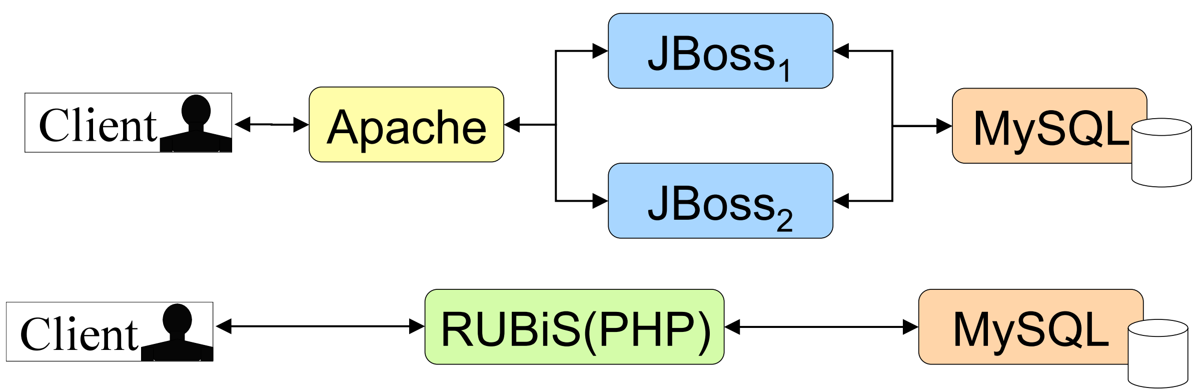
Path Discovery Algorithm

vPath Prototype Components

- Components
  - Online Monitoring Part
    - System call interception at Xen VMM
      - Xen 3.1.0 for x86 32-bit Architecture
      - Guest Linux kernel 2.6.18
    - Information collection for feeding to the analyzer
  - vPath Log Analyzer
    - Algorithms for preprocessing
    - Path construction logic

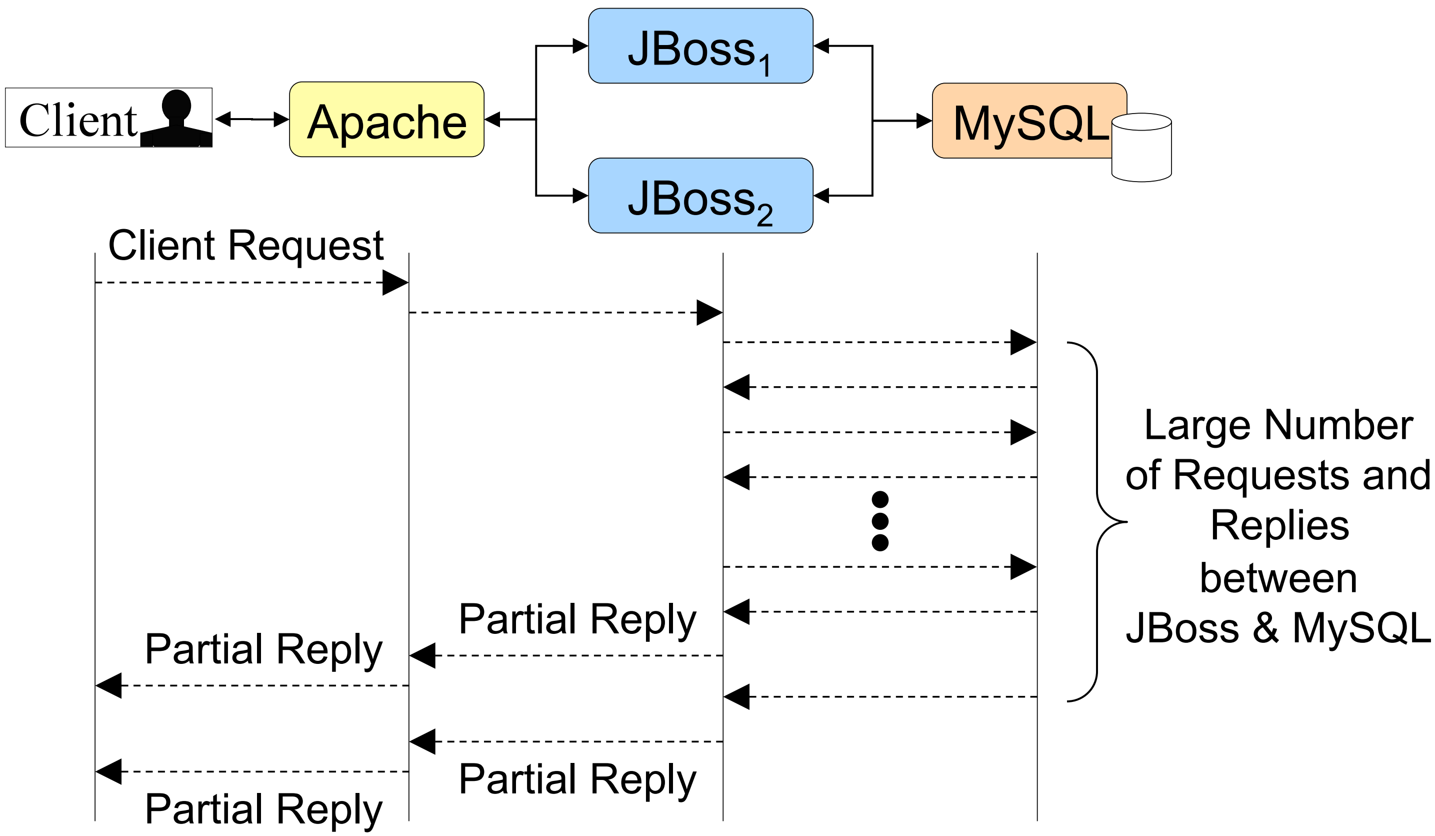
Evaluation Set-up

- Workloads
  - TPC-W – representing Java-based applications
  - RUBiS(PHP version)
  - vApp – custom C socket programming
  - MediaWiki
- System Set-up for TPC-W & RUBiS
  - Separate VMs for each application



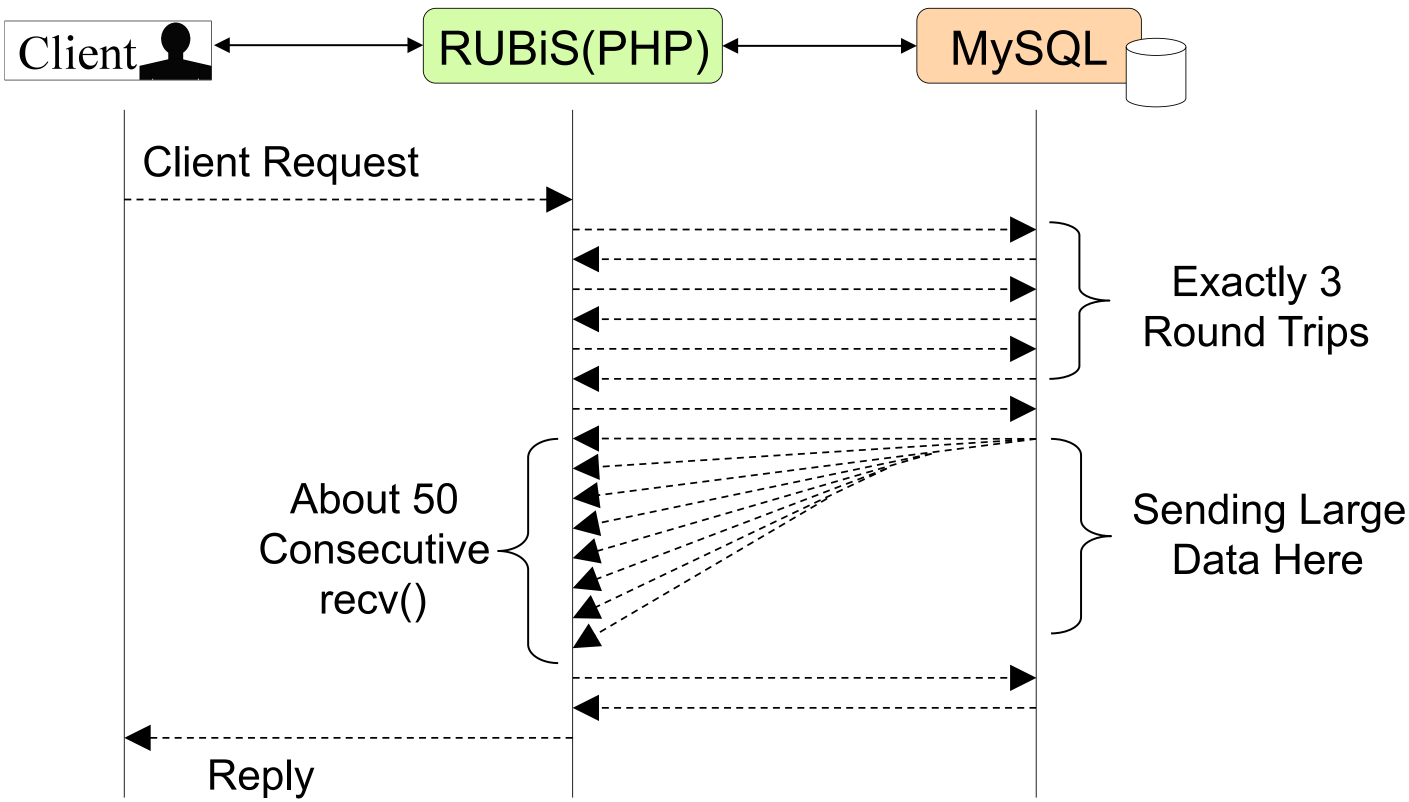
TPC-W

- Discovered Path for TPC-W



RUBiS

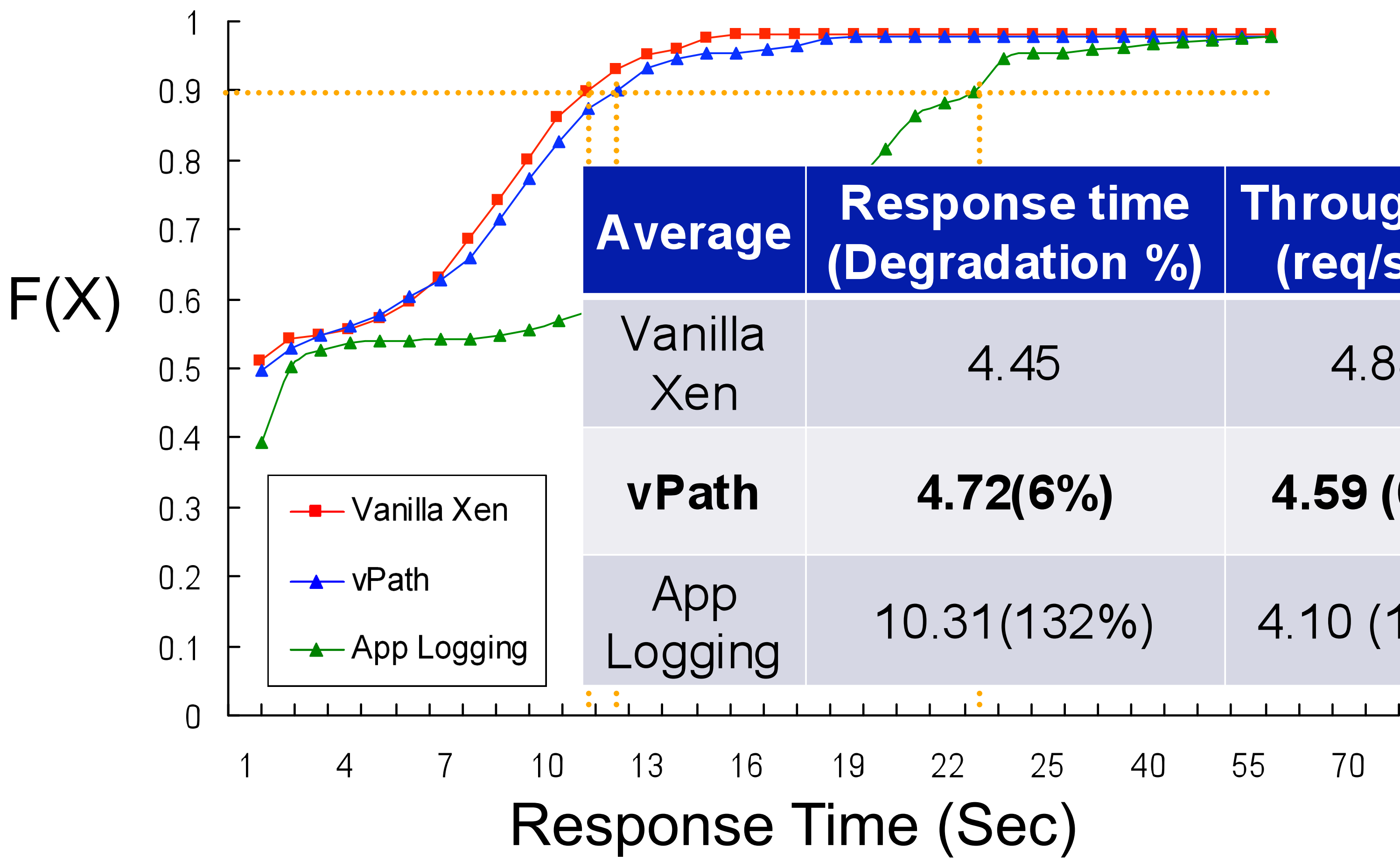
◆ Discovered Path for RUBiS



Overhead

- vPath overhead on TPC-W response time

CDF of TPC-W Response Time

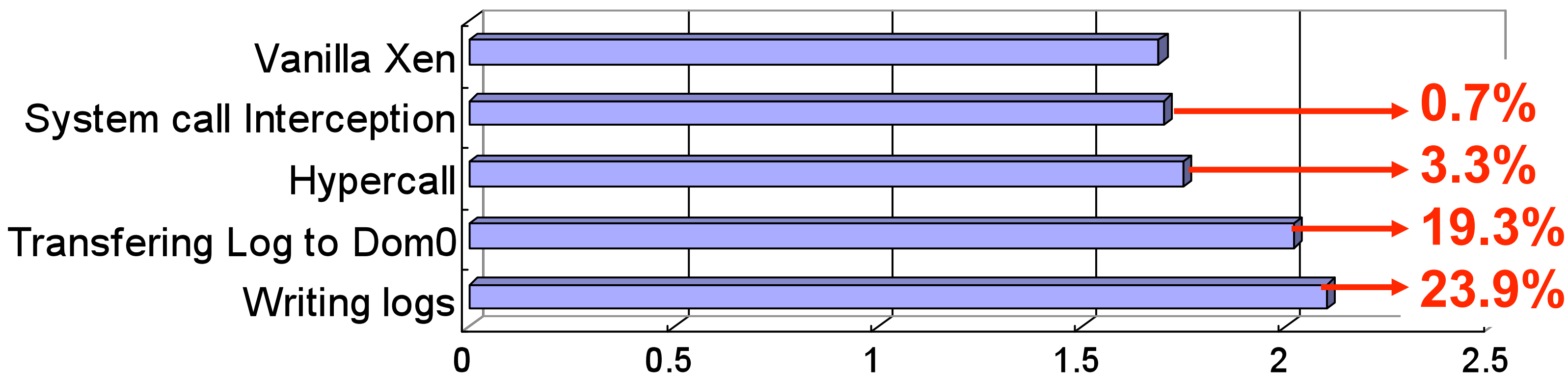


Average	Response time (Degradation %)	Throughput (req/sec)
Vanilla Xen	4.45	4.88
vPath	**4.72(6%)**	**4.59 (6%)**
App Logging	10.31(132%)	4.10 (16%)

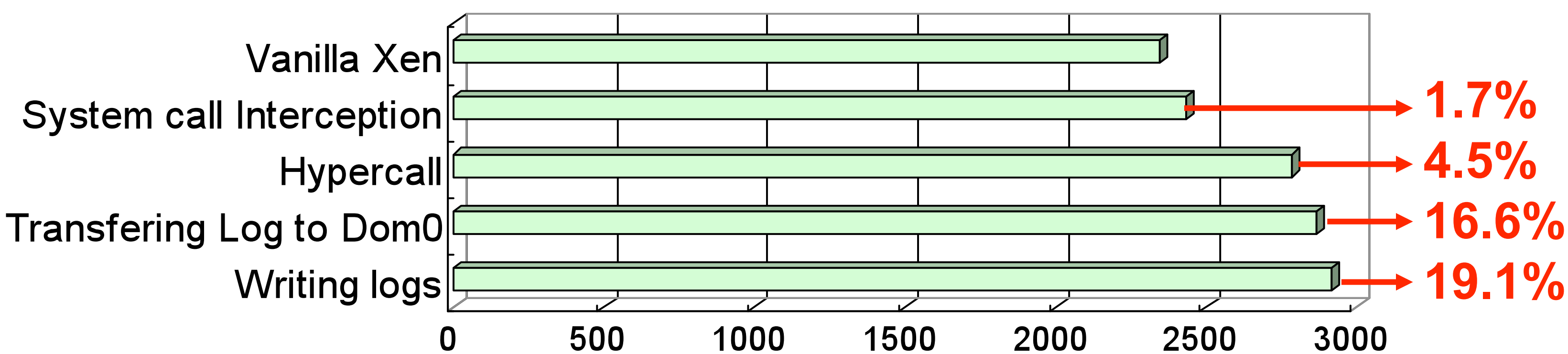
Dissection of vPath Overhead♪

- Worst case overhead measurement

Response Time (in Sec)



Throughput (in req/sec)



Limitations

- vPath works for Multi-threaded model
  - Unable to apply to event-driven or SEDA model
    - We argue that multi-threaded model is dominant
- Accesing socket information
  - Current implementation uses hypercall
    - Modification of the para-virtualized guest VM
    - Each system call incurs another mode-switch

Conclusion and Future work

- Proposal of vPath technique
  - Accurate and non-intrusive technique of path dis covery in a virtualized environment
  - vPath exploits multi-threaded nature of applicatio ns and communication patterns
  - Low run-time overhead
- Future Work
  - Implementation of pure VMM-based approach
  - More study on behaviors of various apps
  - Collecting resource consumptions per path

Thank you