Shifting GEARS to Enable Guest-context Virtual Services

September 18, 2012

Kyle C. Hale

Lei Xia

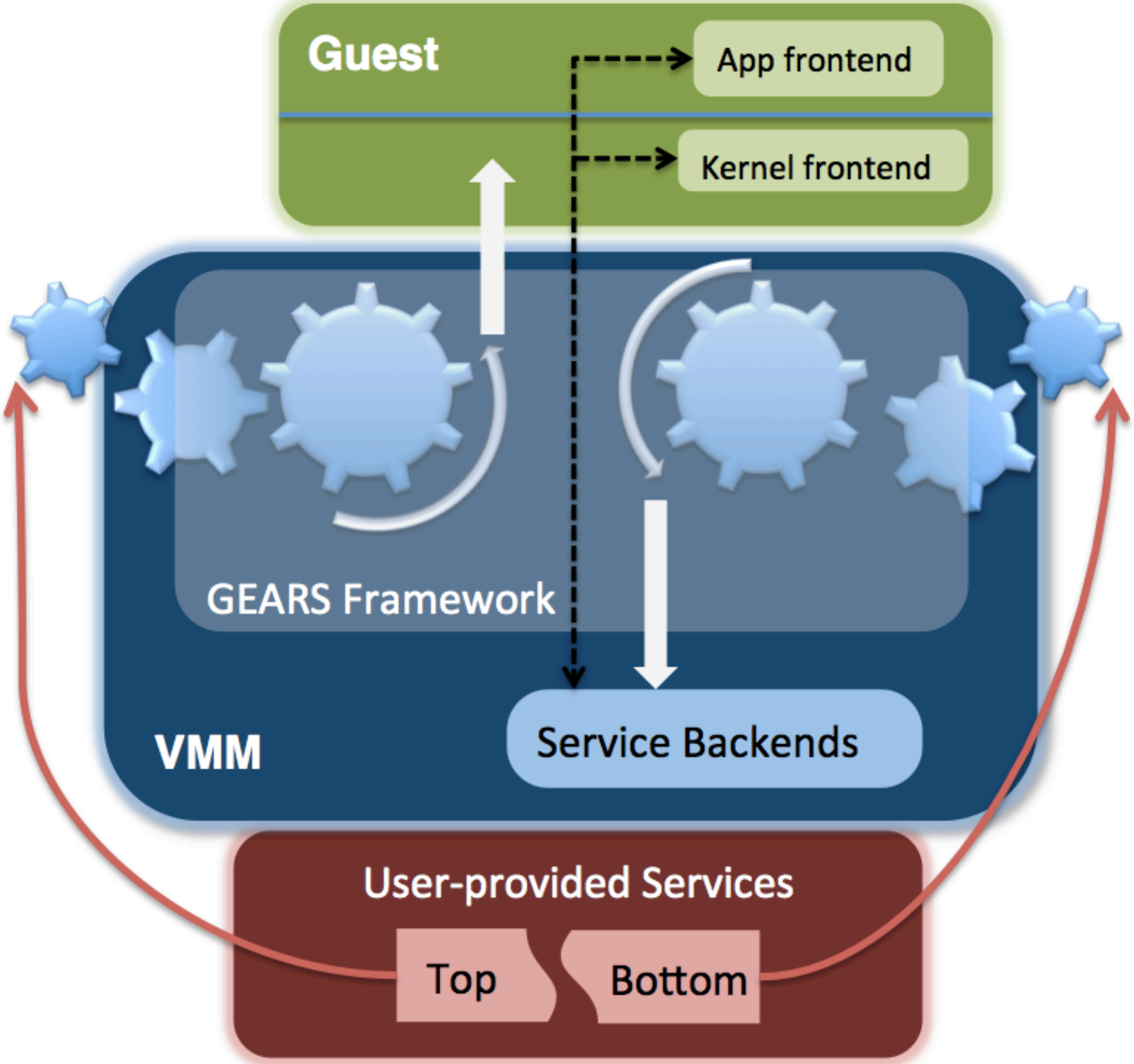
Peter Dinda

EECS Department
Northwestern University

http://v3vee.org

OVERVIEW

- We advocate hoisting implementations of VMM services up into the guest without guest cooperation
- GEARS (Guest Examination And Revision Services): framework for guest-context virtual services
- Allows easy development of services, with potential performance gains and small increase in VMM complexity
- Two prototype guest-context virtual services
  - Overlay networking accelerator (latency decrease by 3-20%)
  - MPI Accelerator (native memcopy bandwidth for colocated VMs)



GEARS

Operation

OUTLINE

- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions

MOTIVATION

- VMM code running within the guest can be simpler, operates at a higher semantic level
- Overheads from VMM exits are substantial
- Allows new classes of services that wouldn’t be possible
- Alternatives, e.g. paravirtualization, symbiotic virtualization, require guest cooperation
- Need a bidirectional interface between VMM and guest, no guest cooperation

OUTLINE

- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions

PALACIOS VMM

- OS-independent, embeddable VMM
- Support for multiple host OSes (Linux, Kitten LWK)
- Open source, available at

  http://v3vee.org/palacios

Palacios

An OS Independent Embeddable VMM

GEARS

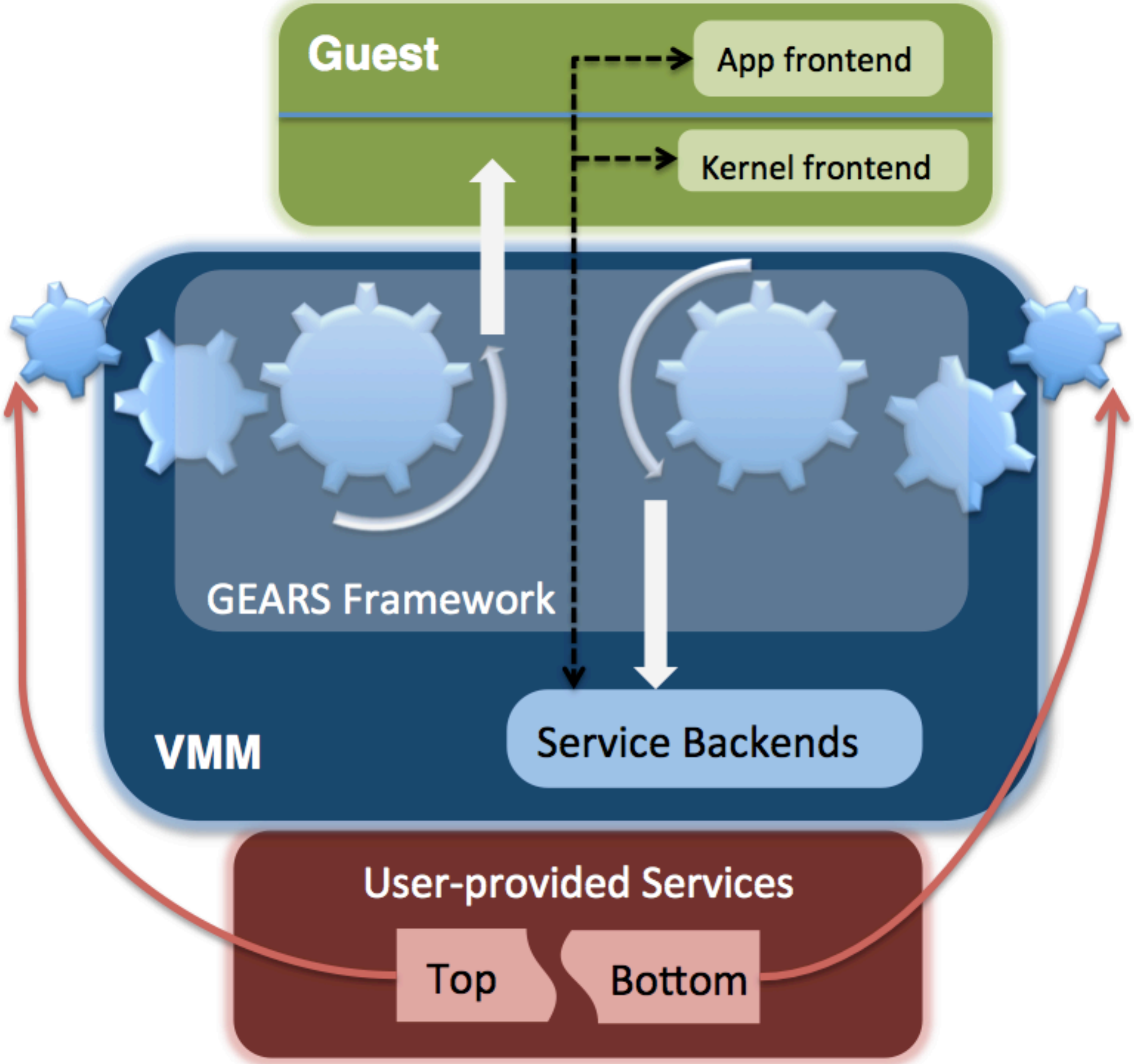
- We claim that to enable wide range of services, need 3 major tools
  - System call interception: track userspace events
  - Process environment modification: pass info to processes
  - Code injection: run VMM code in guest (app and kernel)

- These tools could be built in any VMM, and require little implementation effort

GEARS DETAILS

- Adds little complexity to VMM codebase
- Service developer provides implementations and GEARS transforms and places them appropriately in guest

Component	SLOC
System Call Interception	833
Environment Modification	683
Code Injection	915
Total	2431



GEARS

Operation

SYSCALL INTERCEPTION

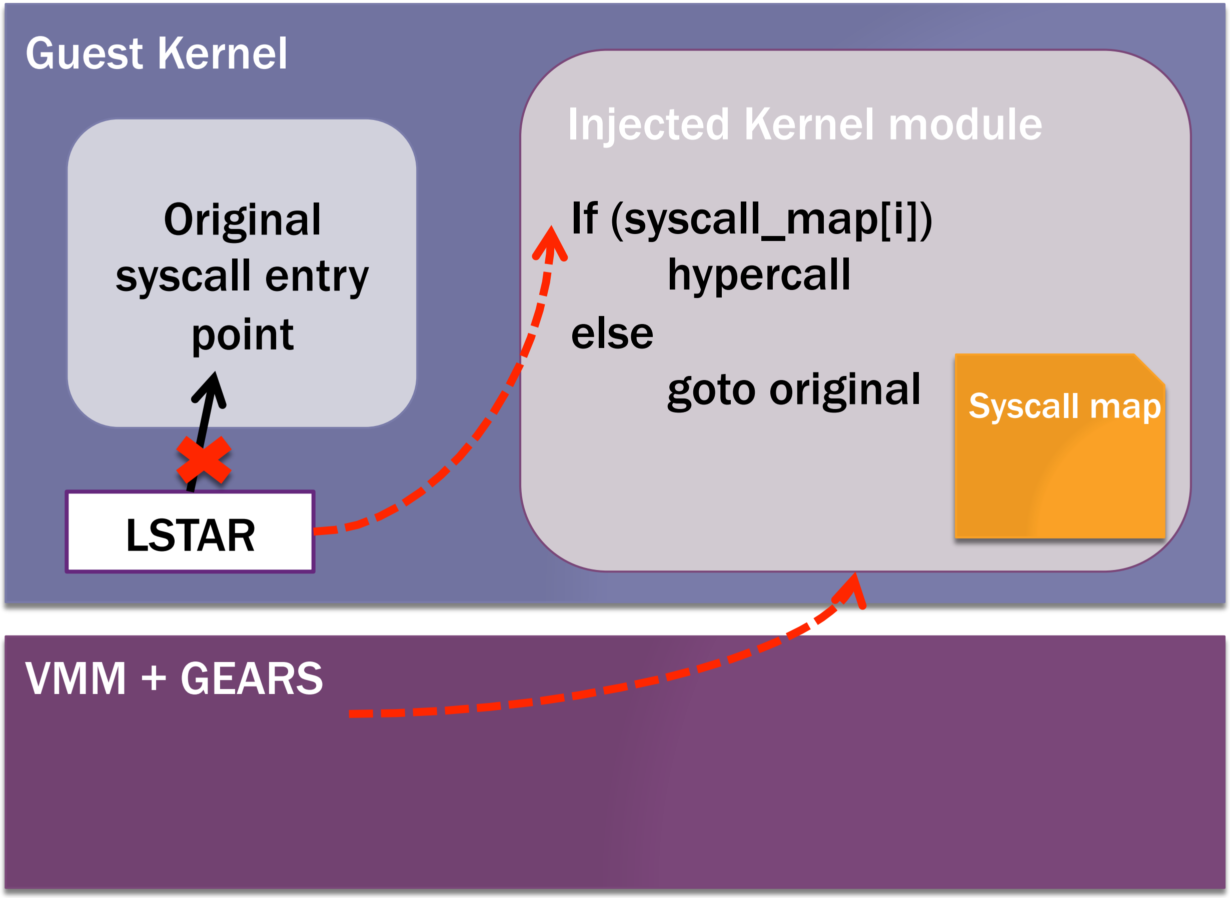
- Introduce system calls as exceptional events to VMM
  - SYSCALL/INT 0x80

- Can build several services on top of this technique
  - Sanity check args against errors/attack
  - Match system call patterns to higher level events

- Used in GEARS to track user-space events at a fine granularity

- Either exit on all syscalls or be selective (requires injected module)

SELECTIVE SYSTEM CALL EXITING



Guest Kernel

Original syscall entry point

LSTAR

Injected Kernel module

```
If (syscall_map[i])
        hypercall
else
        goto original
```

Syscall map

VMM + GEARS

PROCESS ENVIRONMENT MODIFICATION

- Intercept calls to *execve()* to track process creations

- Interception happens before new address space created

- Modify environment variables passed to child process

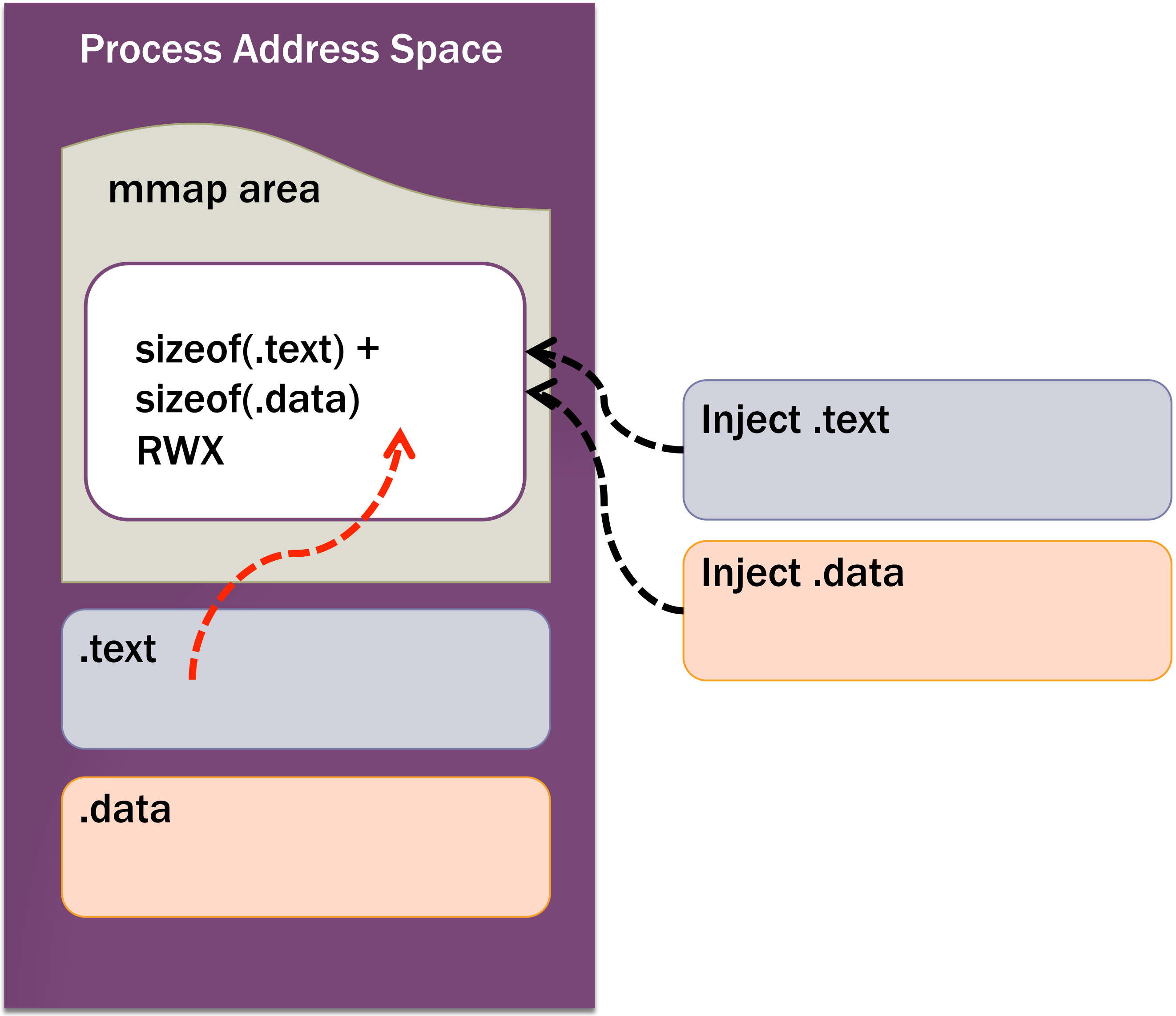
- A few interesting env. vars we can manipulate from VMM
  - LD_PRELOAD
  - LD_BIND_NOW
  - LD_LIBRARY_PATH

- We use LD_PRELOAD in our examples

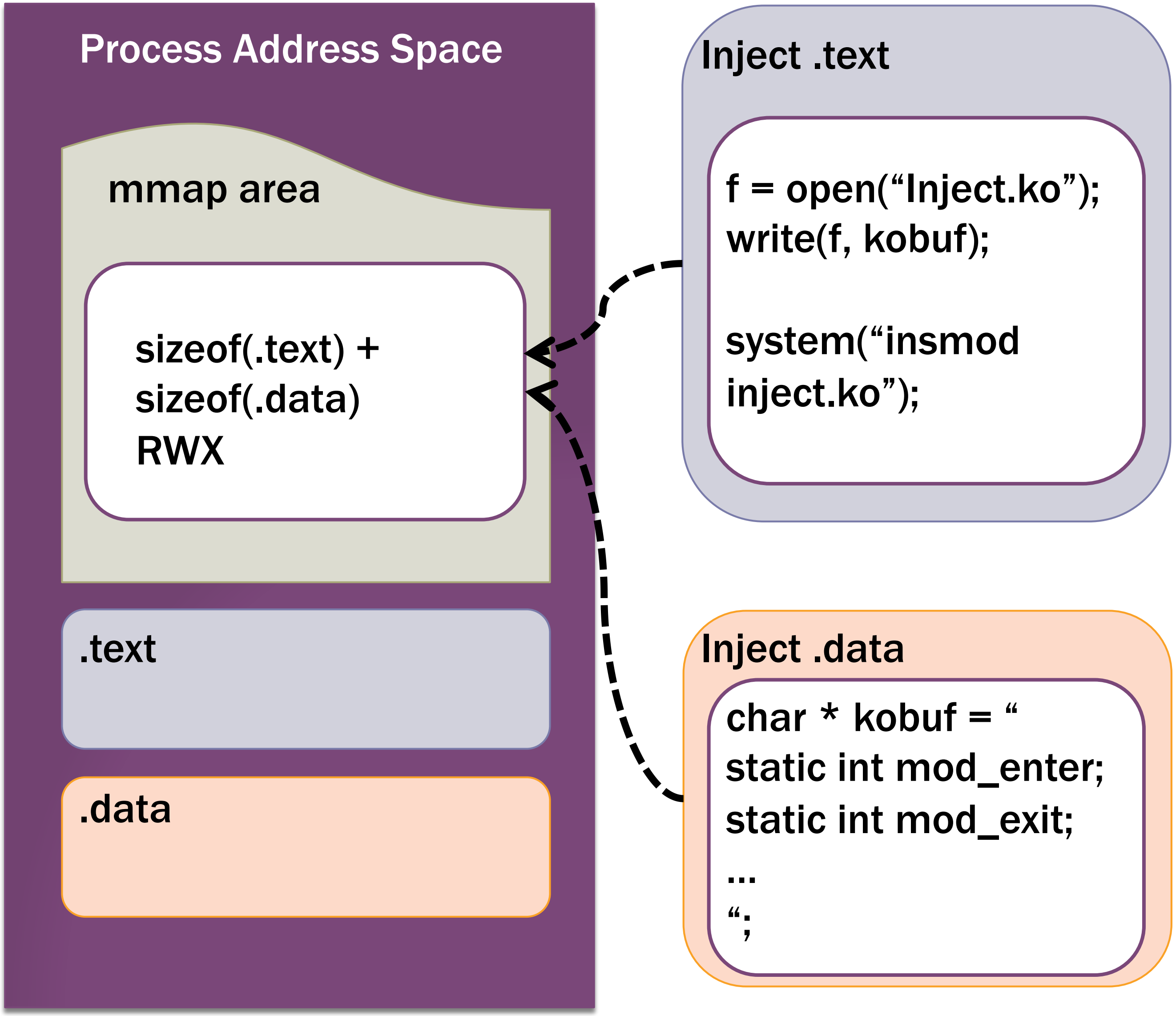
CODE INJECTION

- Allows VMM to run arbitrary code in guest without cooperation
- Core tool for guest-context virtual services
- Userspace injection: map trusted code into process addr. space
- Kernel: use userspace injection to inject kernel module in guest
- Code can be called directly by VMM, or redirect function calls by patching binary

USERSPACE CODE INJECTION



KERNEL CODE INJECTION



Process Address Space

mmap area

sizeof(.text) +
sizeof(.data)
RWX

.text

.data

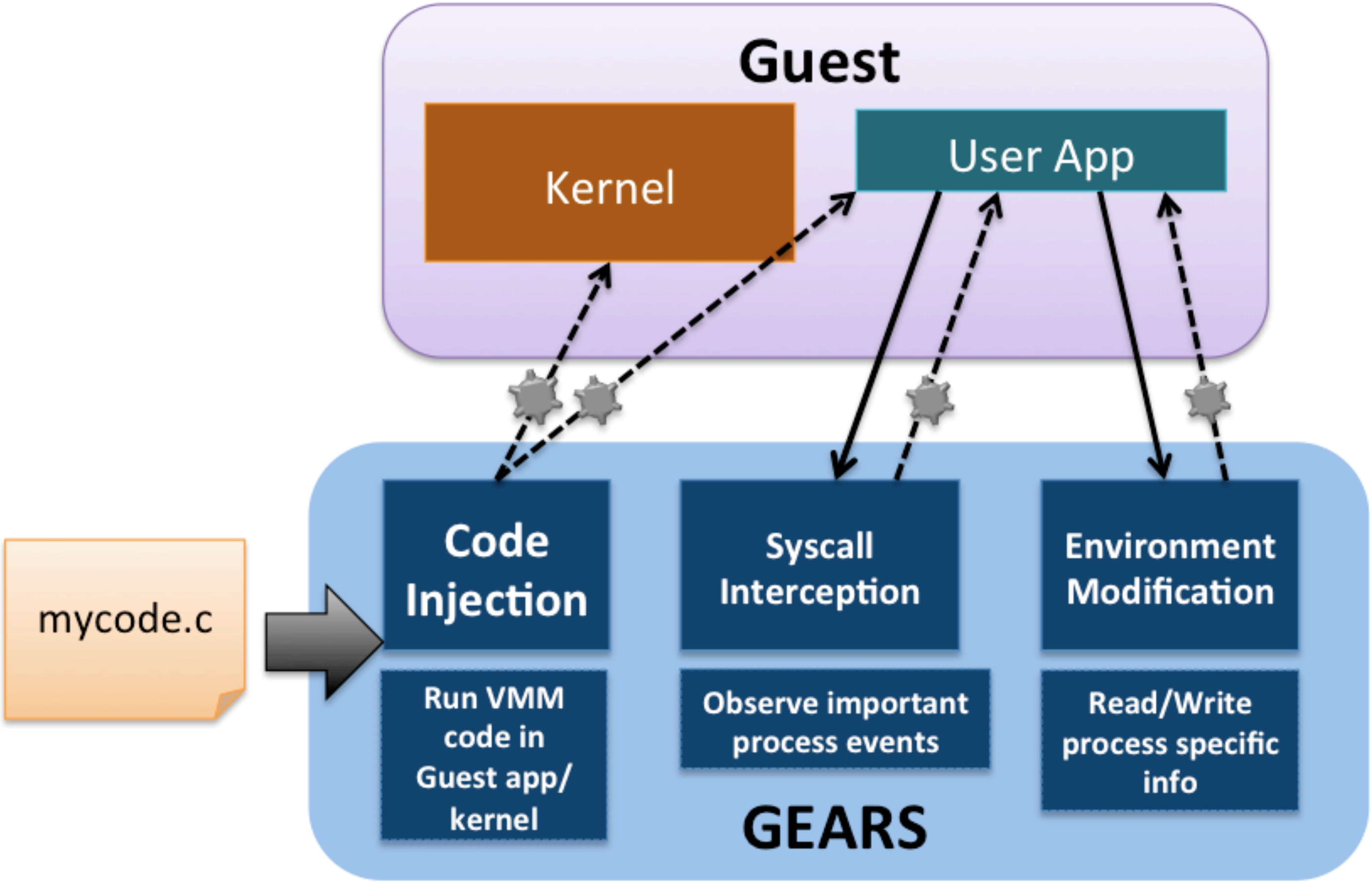
Inject .text

```
f = open(“Inject.ko”);
write(f, kobuf);

system(“insmod
inject.ko”);
```

Inject .data

```
char * kobuf = “
static int mod_enter;
static int mod_exit;
…
“;
```


Guest
Kernel
User App
mycode.c
Code Injection
Syscall Interception
Environment Modification
Run VMM code in Guest app/ kernel
Observe important process events
Read/Write process specific info
GEARS

Service Interaction

OUTLINE

- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions

SYSCALL INTERCEPT LATENCY LOW

getpid() system call

Legacy System Call (INT 0x80)

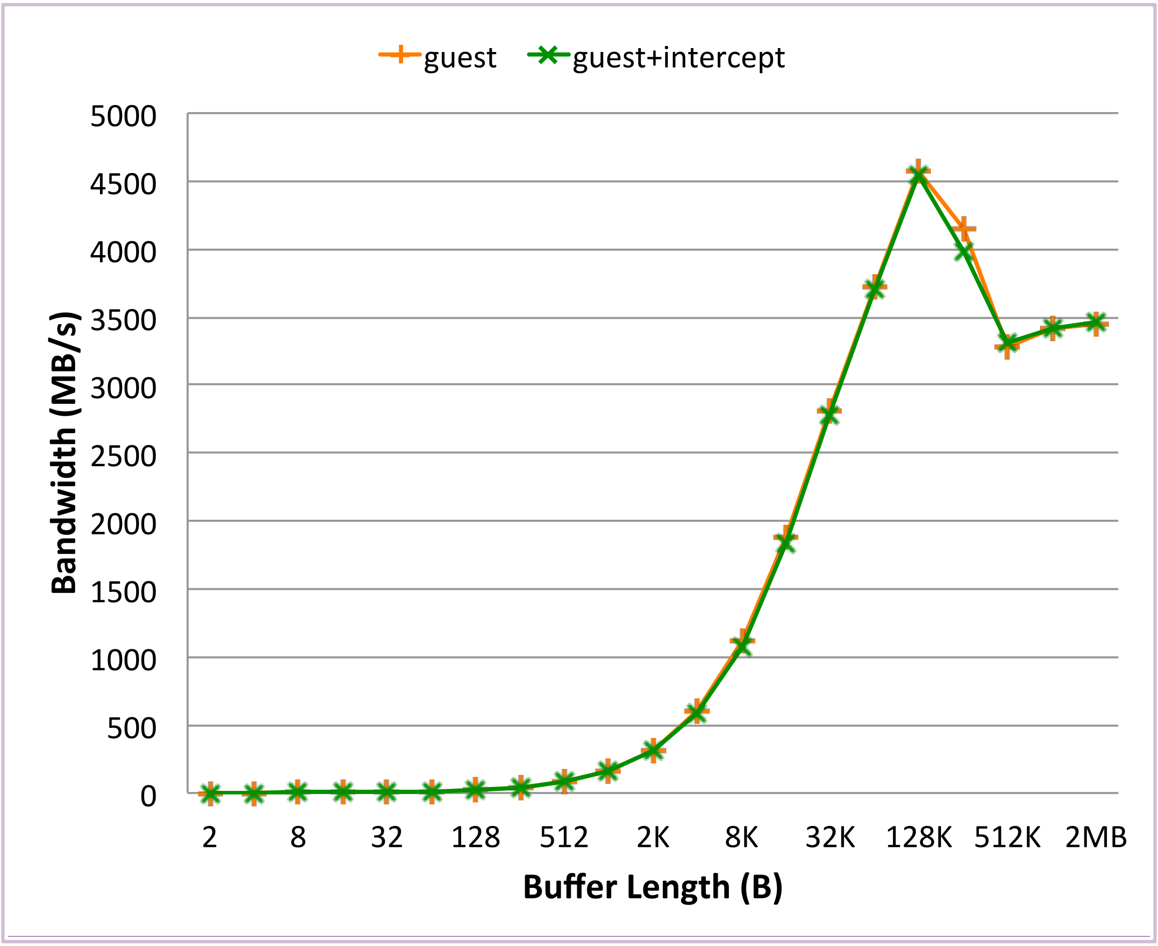
Strategy	Latency (µs)
Guest	4.83
Guest + intercept	10.24

SYSCALL Instruction

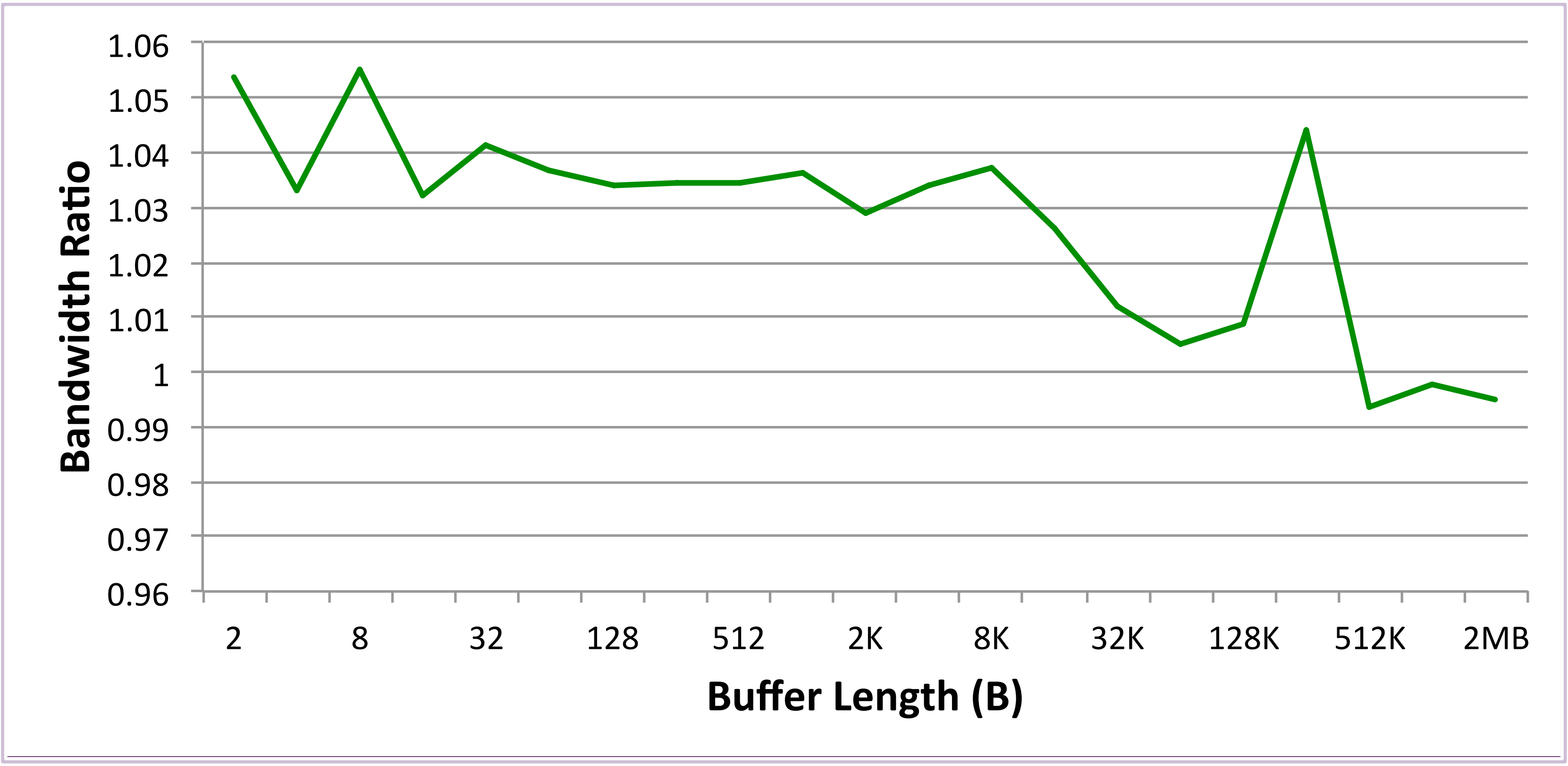
Strategy	Latency (µs)
Guest	4.26
Guest + intercept	4.51

Setup: AMD x86_64, 2.3 GHz Quad-core Opterons
Host: Fedora 15, Linux 2.6.42 Guest: Linux 2.6.38

SYSCALL BANDWIDTH UNCHANGED W/ INTERCEPT



BANDWIDTH RATIO WITH/WITHOUT INTERCEPT



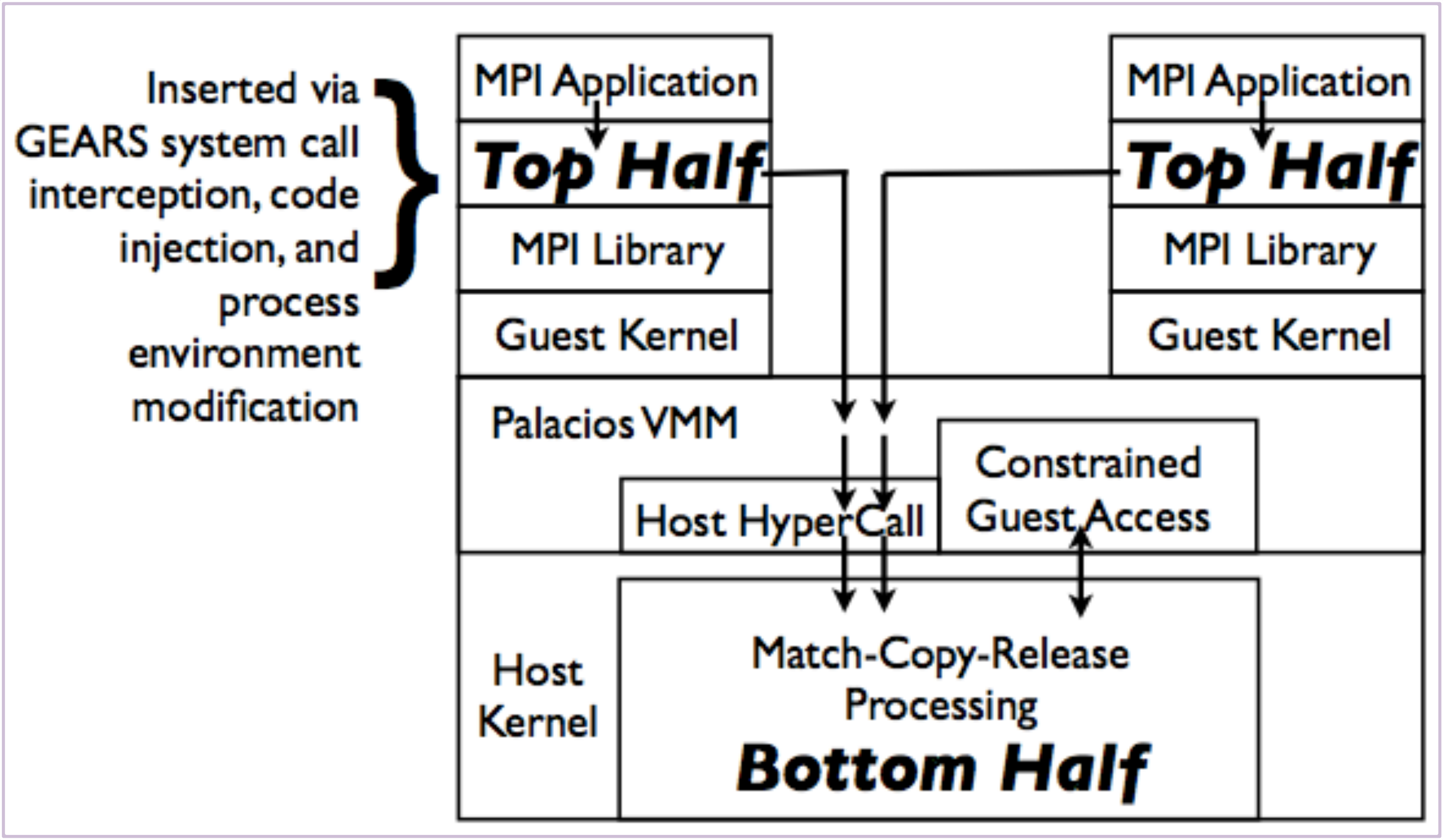
OUTLINE

- Overview
- Motivation
- GEARS
- Evaluation of Tools
- Example Service
- Conclusions

MPI ACCELERATOR

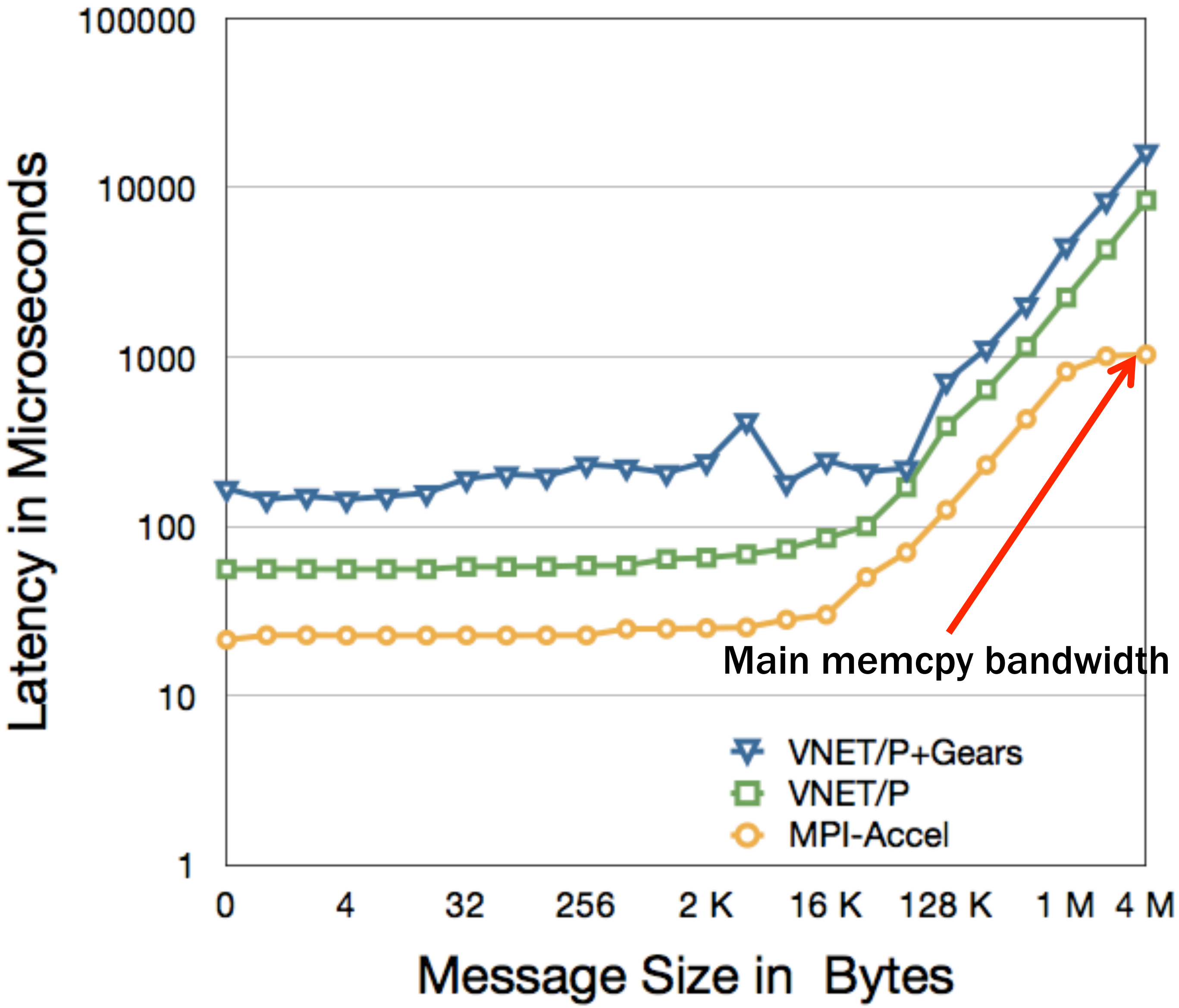
- MPI library in guest is oblivious to VMs on same host
- Use GEARS to transform MPI_Send/Recv (library calls) into memcopy operations
- Building within VMM is difficult because we lose MPI semantics
  - Discern semantics *from guest app*
- Uses userspace code injection and process environment modification

MPI ACCELERATOR



MPI ACCELERATOR

- We focus on blocking send and recv
- Injected library redirects some MPI calls as VMM hypercalls
- Bottom half tracks MPI processes using a tuple (VM ID, virtual core, CR3, executable name)



MPI Accelerator Approaches Main memory copy bandwidth

SERVICE IMPLEMENTATION COMPLEXITY LOW

MPI Accelerator

Component	SLOC
Preload Wrapper (Top Half)	345
Kernel Module (Bottom Half)	676
Total	1021

Overlay Accelerator

Component	SLOC
Vnet-virtio kernel module (Top Half)	329
Vnet bridge (Bottom Half)	150
Total	479

- Overview
- Motivation
- GEARS
- Tools Evaluation
- Example Service
- Conclusions

OUTLINE

CONCLUSIONS

- GEARS, a set of tools to enable guest context virtual services
- Tools that comprise GEARS are few and compact, could be implemented in other VMs
- Developers, with little knowledge of VMM core, and without modifying guest, can use GEARS to build virtual services that are
  - smaller
  - faster
  - easier to understand
  - otherwise unfeasible

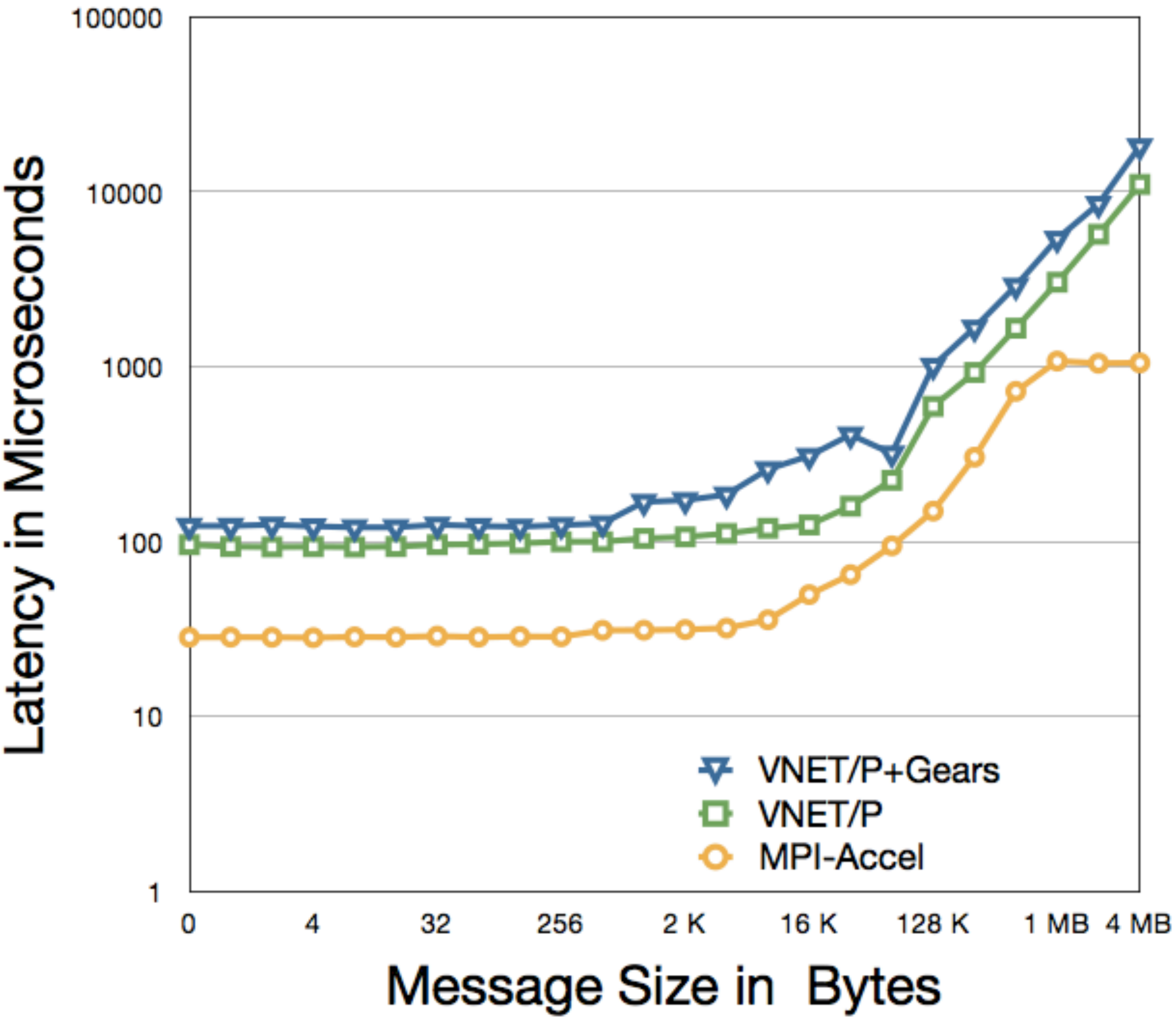
FUTURE WORK

- Explore boundaries between VMM-injected code and guest code
- Safely run trusted components in guest, give them privileged HW access
- Application-specific VMM awareness, guest context virtual services as an alternative to OS ABI

QUESTIONS?

- Get Palacios (with GEARS) online
  http://v3vee.org/palacios
- Kyle Hale: http://users.eecs.northwestern.edu/~kch479

Palacios
An OS Independent Embeddable VMM

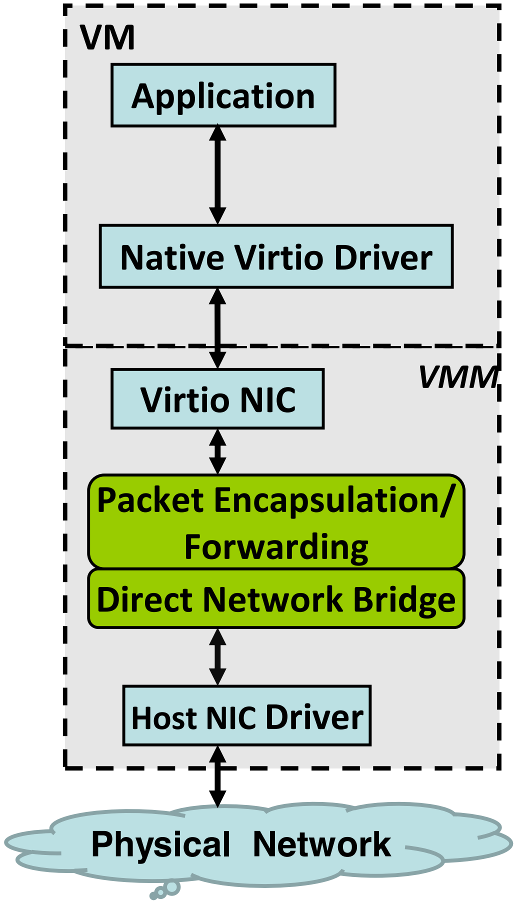


ACCELERATOR PERFORMANCE

Lewinsky

OVERLAY ACCELERATOR

- VNET/P: overlay networking system in Palacios
  - Layer 2 abstraction
  - Near native performance in 1Gbps/10Gbps
  - 75% native throughput, 3-5x native latency fully encapsulated
- Overheads due to VM exits, data copies, data ownership xfer
- Use GEARS to move part of datapath into guest



VM

Application

Native Virtio Driver

VMM

Virtio NIC

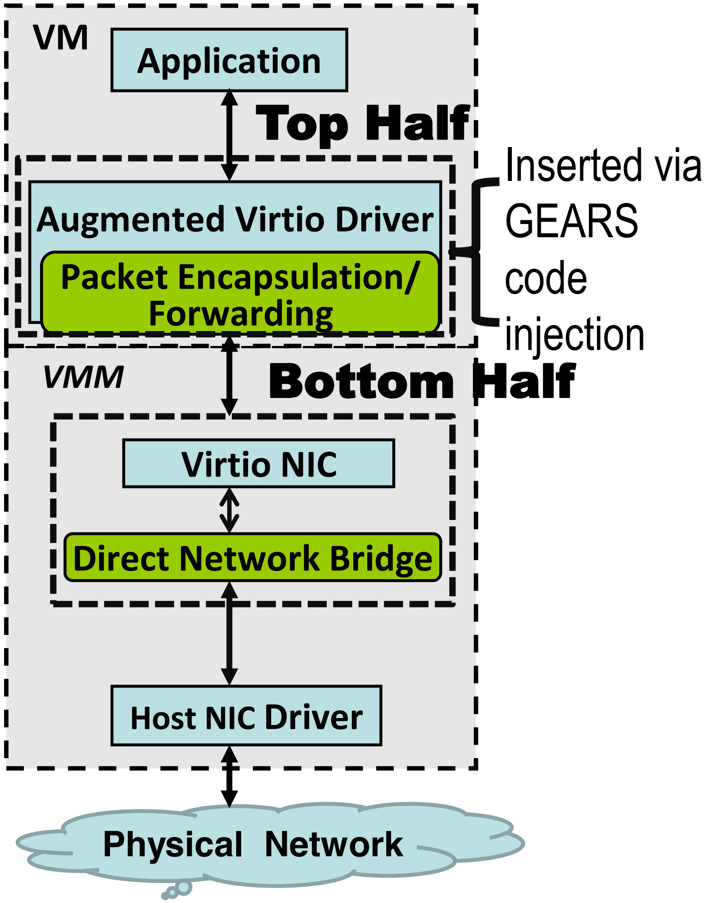
Packet Encapsulation/ Forwarding

Direct Network Bridge

Host NIC Driver

Physical Network

VNET/P



VM

Application

Top Half

Augmented Virtio Driver

Packet Encapsulation/ Forwarding

Inserted via GEARS code injection

VMM

Bottom Half

Virtio NIC

Direct Network Bridge

Host NIC Driver

Physical Network

VNET/P Accelerator

VNET Accelerator

VNET ACCELERATOR: SMALL LATENCY IMPROVEMENT

Benchmark	Native	VNET/P	VNET/P Accel
Latency			
min	0.082 ms	0.255 ms	0.205 ms
avg	0.204 ms	0.475 ms	0.459 ms
max	0.403 ms	2.787 ms	2.571 ms
Throughput			
UDP	922 Mbps	901 Mbps	905 Mbps
TCP	920 Mbps	890 Mbps	898 Mbps

- Proof of concept
- Could improve further with more functionality in guest (with privileged HW access)

WHY CODE INJECTION?

- i.e., why do we care whether a guest cooperates?

- What about a compromised guest? (VMM could forcefully repair a guest)
- What about guests where you don’t have control, but want to enforce some invariant?
- What if changes need to be made on the fly? E.g. host-guest file copy=>saved time

WHAT ABOUT SECURITY?

- We’ve essentially increased the possible number of attack vectors into the hypervisor, right?

- True, but there may be ways we can protect guest-context VMM code better than other interfaces (e.g. Secure in-VM monitoring)

- VMM can remove its code from the guest, lock down the guest etc. when a vulnerability is found.