Performance Implications of Anti-Virus Execution on a Virtual Platform

Derek Uluski
Micha Moffie
David Kaeli

Institute for Information Assurance @ NU

Anti-Virus Security Mechanisms

- Signature Matching
  - Program will refer to a dictionary of “signatures” or sequences of code known to be part of a malicious file
  - If a signature is found in the file in question, it is marked as a virus
  - Disadvantages:
    - Requires continuous updates
    - Cannot detect “zero-day attacks”
- Heuristics
  - Set of rules that program will apply to detection mechanism
    - For example, if the file contains self-modifying code
  - If the file in question violates any of the given rules, it is marked as a virus
  - Disadvantages:
    - Generates false positives

Anti-Virus Security Mechanisms

- Sandboxing
  - Program will emulate the operating system environment
  - Monitor and analyze program execution
  - After completed, the resulting environment is analyzed for changes indicating the presence of a virus
  - Disadvantages:
    - Consumes significant amount of resources
    - Consequently, cannot be run “on-access”

A/V Characterization Studies

- AVTest.org 2002 Windows XP Test
  - System Degradation, copying a file (based on time trial – relative time increase consumed compared to copy without anti-virus software present)
    - F-Prot: 54%
    - Norton: 118%
    - McAfee: 77%
    - PC-Cillin: 34%
    - Kaspersky: 116%
    - AntiVir PE: 51%

A/V Characterization Studies

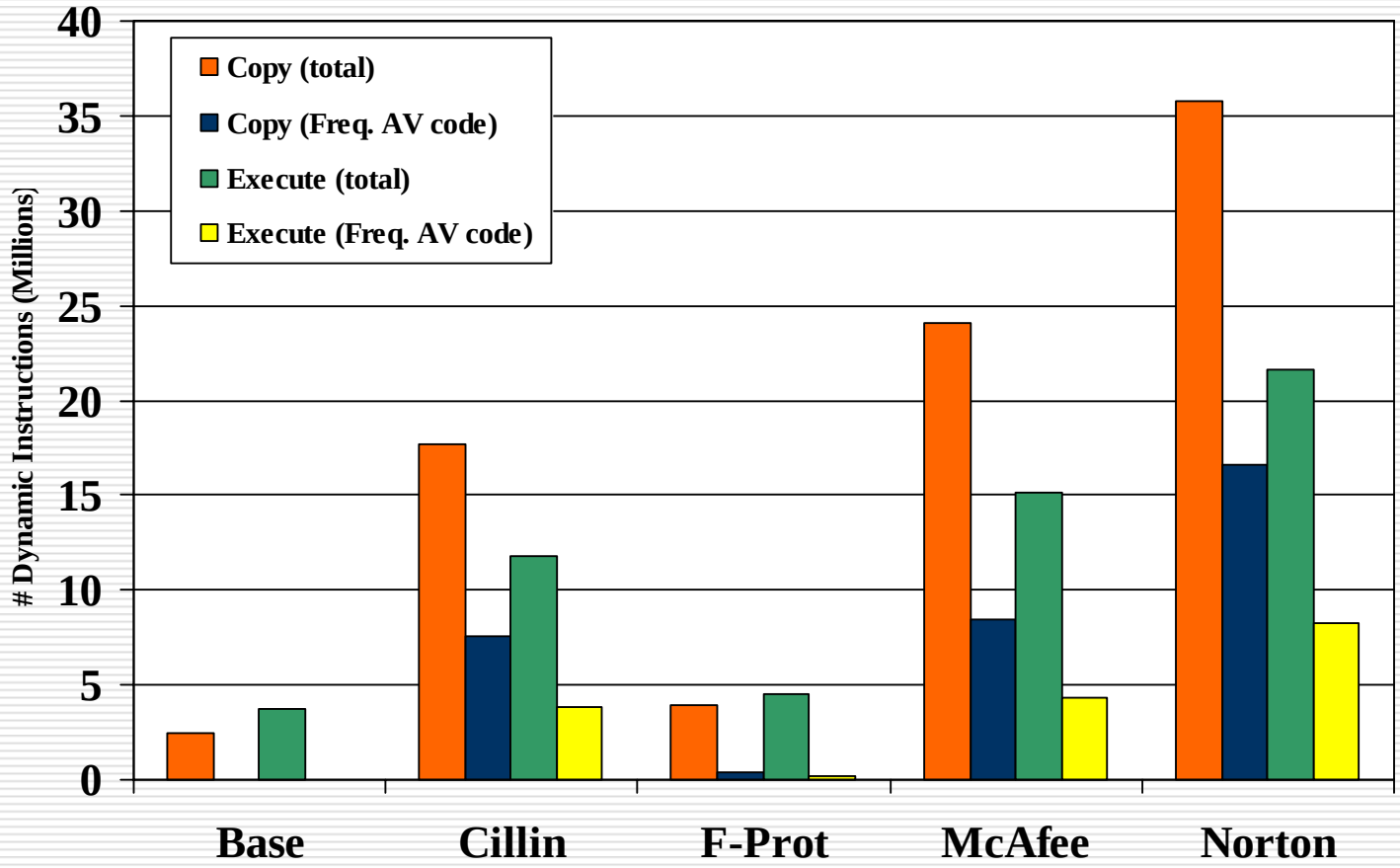
Colby-Sawyer College AV software comparison study

- Included accuracy results from virus bulletin

Package	Memory Consumed While Idle (MB)
Panda	55.5
F-Secure	31.5
Etrust	31
McAfee	31
PC-Cillin	28
RAV	26.5
Norman	26
Kaspersky	23.5
Norton	23
Bit Defender	17
Nod32	13.5
F-Prot	7

Motivation

Copy/Execution of "Hello, world" Application



Outline

- Introduction
- Experimentation Methodology
- Performance Degradation
- Virtualization Layer Proposal
- Conclusions

Introduction

- AV software execution degrades application performance
  - Why?
    - Files need to be scanned for specific sequences
      - instruction sequences in a executable binary
      - VB code sequences in Office documents
    - Not only generates many extra instructions, but also places pressure on the memory system
    - On access vs. on demand

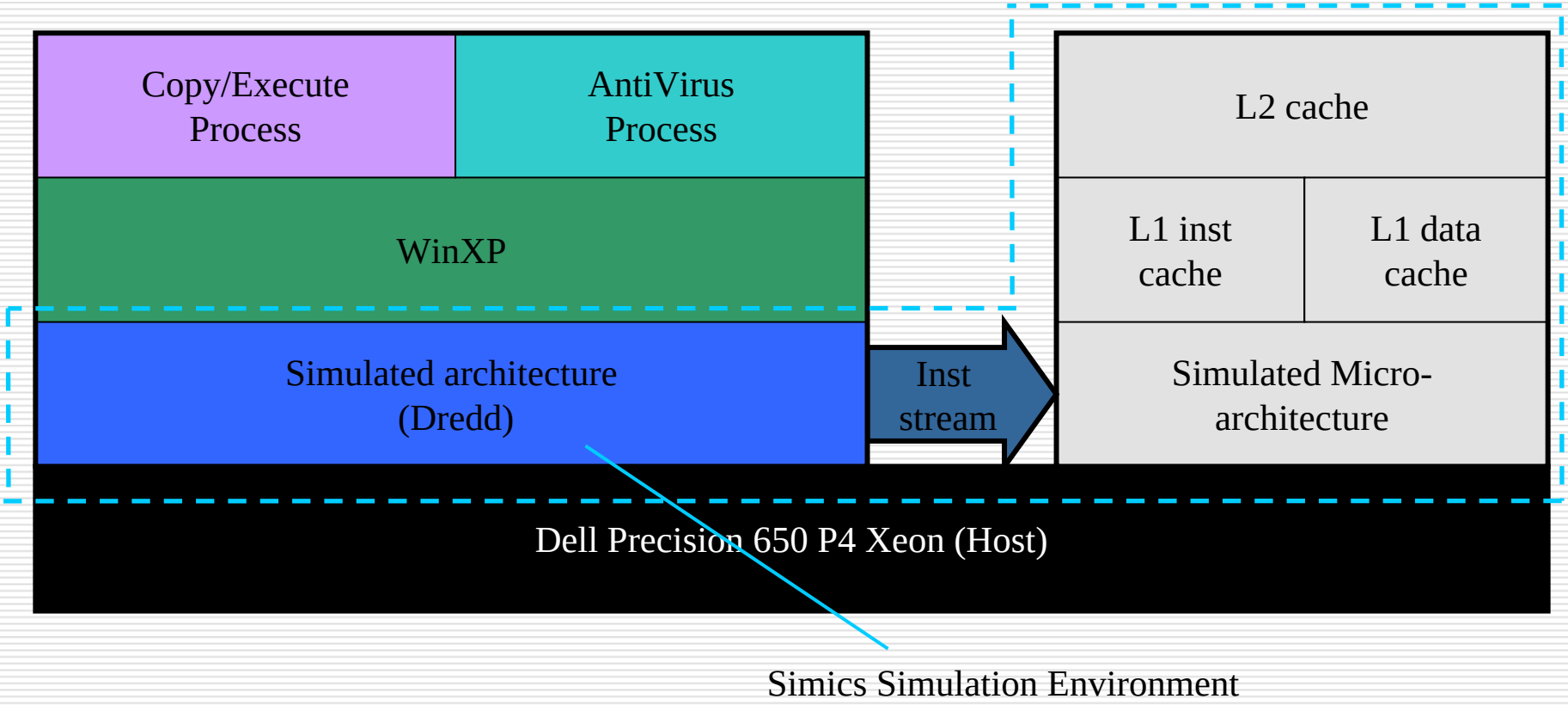
Experimentation Methodology

- Virtutech Simics
  - Architecture simulator for uniprocessor and multiprocessor machines
  - System-level instruction set simulator
    - Models interfaces to buses, interrupt controllers, disks, and video controllers
    - Models in-order CPU instruction execution

Architectural Model

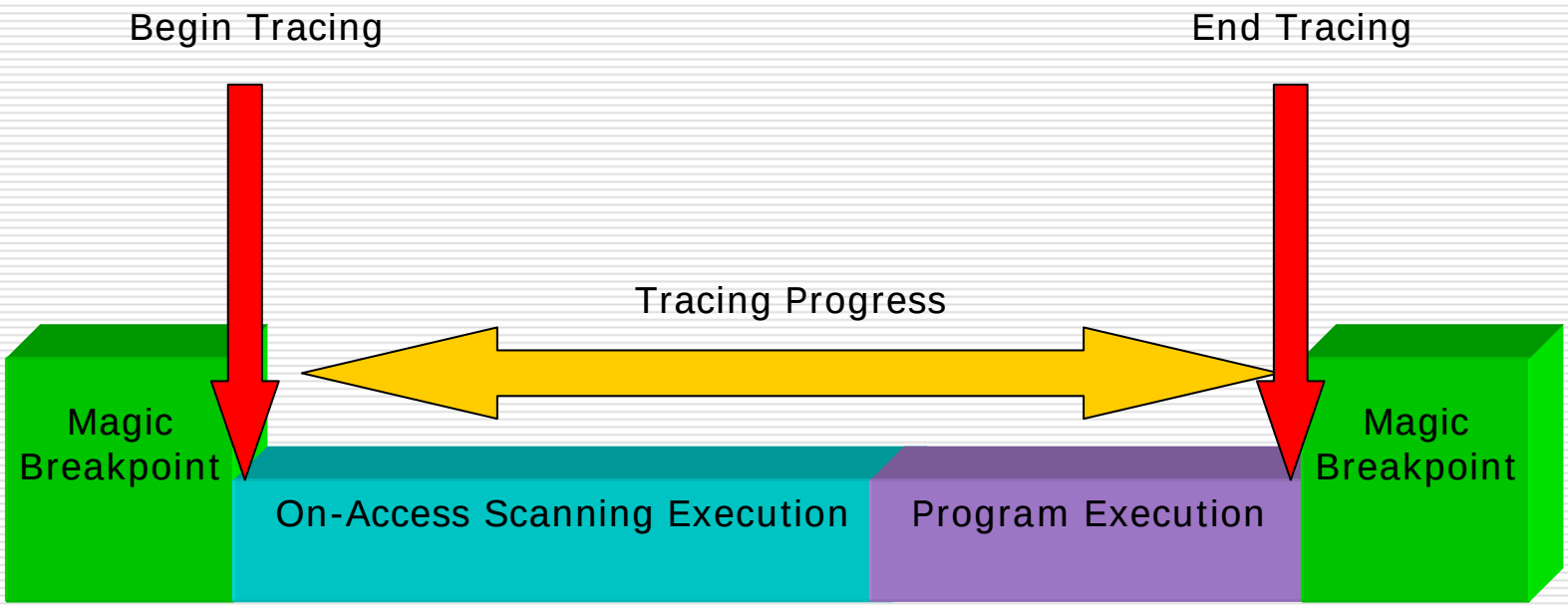
Processor	Intel Pentium 4 2.0A
Operating Frequency	2 GHz
L1 Ins. Cache	12 KB
L1 Data Cache	8 KB
L2 Cache	512 KB
Memory	256 MB

Experimentation Methodology



Experimentation Methodology

- Methodology



Experimentation Methodology

- Custom Tracing Tools
  - Print-instruction: sequentially decoded instructions
  - Va-profile: frequency of each virtual address accessed
  - Inst-mix: frequency of decoded instructions
  - Print-data: sequentially accessed data addresses
  - Data-pattern: starting address with linear data accesses
  - Timer: executed instruction counter
  - Opcode: sequential VA and instruction opcode
  - Pid-Trace: Same as opcode with PID tag
  - Data-access: number of memory transactions resulting from the execution of an instruction @ given virtual address

Example

- Postprocessing: Determining *Hot* code
  - Collect frequencies of virtual addresses (using *va-profile* tool) and examine potential blocks
    - In given example, we see 3 different instruction sequences that may be hot code
    - Note: high virtual addresses help to identify DLL or system service

VA	Freq.
0xf6a13e00	94927
0xf6a13dfd	94927
0xf6a13df9	94927
0xf6a13df7	94927
0xf6a13df5	94927
0xf6a13df3	94927
0xf6a13df0	94927
0xf6a13de9	94927
0xf6a13de6	94927
0xf6a0ee2c	94255
0xf6a0ee29	94255
0xf6a0ee27	94255
0xf6a0ee24	94255
0xf6a0ee36	94236
0xf6a0ee33	94236
0xf6a0ee31	94236
0xf6a0ee2e	94236

Example

- Drill-Down
  - Take a closer look at potential block by using breakpoints and trace module

```
simics> break 0xf6a13de6
Breakpoint 1 set on address 0xf6a13de6 with access mode 'x'
simics> run-command-file file = scripts\intro_overhead_test\intro_overhead_test.simics
cdrom 'intro_test' created
Ejecting media from CD-ROM drive
Inserting media 'intro_test' in CD-ROM drive
Code breakpoint 1 reached.
[cpu0] <cs:0xf6a13de6> <p:0x0b1d3de6> 0f b7 f8              movzx edi, ax
[trace module] Created a trace0 object
Module trace loaded
Tracing enabled
inst: [        1] CPU  0 <cs:0xf6a13de6> <l:0xf6a13de6> <p:0x0b1d3de6> 0f b7 f8              movzx edi, ax
inst: [        2] CPU  0 <cs:0xf6a13de9> <l:0xf6a13de9> <p:0x0b1d3de9> 0f af ba fc 0c 00 00  imul edi,dword ptr 0xcfc[edx]
data: [        1] CPU  0 <ds:0xe1c16d04> <l:0xe1c16d04> <p:0x0b4c3d04> Vani WB Read   4 bytes  0x8
data: [        2] CPU  0 <ds:0xe1c16d04> <l:0xe1c16d04> <p:0x0b4c3d04> Vani WB Read   4 bytes  0x8
inst: [        3] CPU  0 <cs:0xf6a13df0> <l:0xf6a13df0> <p:0x0b1d3df0> 8b 5d 10              mov ebx,dword ptr 0x10[ebp]
data: [        3] CPU  0 <ss:0x814b7738> <l:0x814b7738> <p:0x018b7738> Vani WB Read   4 bytes  0x6c5a4d
inst: [        4] CPU  0 <cs:0xf6a13df3> <l:0xf6a13df3> <p:0x0b1d3df3> 03 f9                 add edi,ecx
inst: [        5] CPU  0 <cs:0xf6a13df5> <l:0xf6a13df5> <p:0x0b1d3df5> 3b 1f                 cmp ebx,dword ptr [edi]
data: [        4] CPU  0 <ds:0xe1c81270> <l:0xe1c81270> <p:0x0b4ee270> Vani WB Read   4 bytes  0xf4755a4d
inst: [        6] CPU  0 <cs:0xf6a13df7> <l:0xf6a13df7> <p:0x0b1d3df7> 74 09                 je 0xf6a13e02
inst: [        7] CPU  0 <cs:0xf6a13df9> <l:0xf6a13df9> <p:0x0b1d3df9> 66 8b 47 04           mov ax,word ptr 0x4[edi]
data: [        5] CPU  0 <ds:0xe1c81274> <l:0xe1c81274> <p:0x0b4ee274> Vani WB Read   2 bytes  0x82a
inst: [        8] CPU  0 <cs:0xf6a13dfd> <l:0xf6a13dfd> <p:0x0b1d3dfd> 66 85 c0              test ax,ax
inst: [        9] CPU  0 <cs:0xf6a13e00> <l:0xf6a13e00> <p:0x0b1d3e00> eb e2                 jmp 0xf6a13de4
inst: [       10] CPU  0 <cs:0xf6a13de4> <l:0xf6a13de4> <p:0x0b1d3de4> 74 29                 je 0xf6a13e0f
Code breakpoint 1 reached.
[cpu0] <cs:0xf6a13de6> <p:0x0b1d3de6> 0 0 0 0 0 0 0 0  0f b7 f8              movzx edi, ax
```

Example

- Verification
  - Procedure
    - Verify sequence of code originates from Anti-Virus Binary

```
duluski@barrett ~/C/bins/nav_b
$ ./binary-search 0fb7f80fafbafc0c00008b5d1003f93b1f7409 *.*
Found sequence 0fb7f80fafbafc0c00008b5d1003f93b1f7409 in file navex15.sys
```

    - Repeat search and verification process for all significant possibilities for basic blocks
  - This process is done to ensure that the data collected is from the anti-virus software package and not from external execution streams (such as operating system)

Experimentation Methodology

Anti-Virus Software Packages

- AV software packages evaluated
  - Trend Micro PC-Cillin 11.0.0.1253
  - FRisk F-Prot 3.14b
  - McAFee Anti-Virus 8.0.20
  - Norton Anti-Virus 2004 10.0.0.109
- The most prominent packages in the U.S. market at the beginning of 2004

Experimentation Methodology

Frequent "Hot" Code Examples

PC-Cillin

```
mov edx, dword ptr 0xb0[ebp]
inc ecx
add eax, 0xc
cmp ecx, edx
mov dword ptr 0xd4[ebp], ecx
jl 0xf45cc81a
```

F-Prot

```
mov exc, dword ptr 0x8[ebp]
mov cl, byte ptr[ecx]
cmp cl, byte ptr 0xc[ebp]
je 0xf76a713
```

McAfee

```
xor edi, edi
mov ecx dword ptr 0x8[ebp]
mov al, byte ptr 0x1[ebx]
lea edx,[edi][ecx]
mov cl, byte ptr [edi][ecx]
cmp al, cl
jne 0x1203c028
```

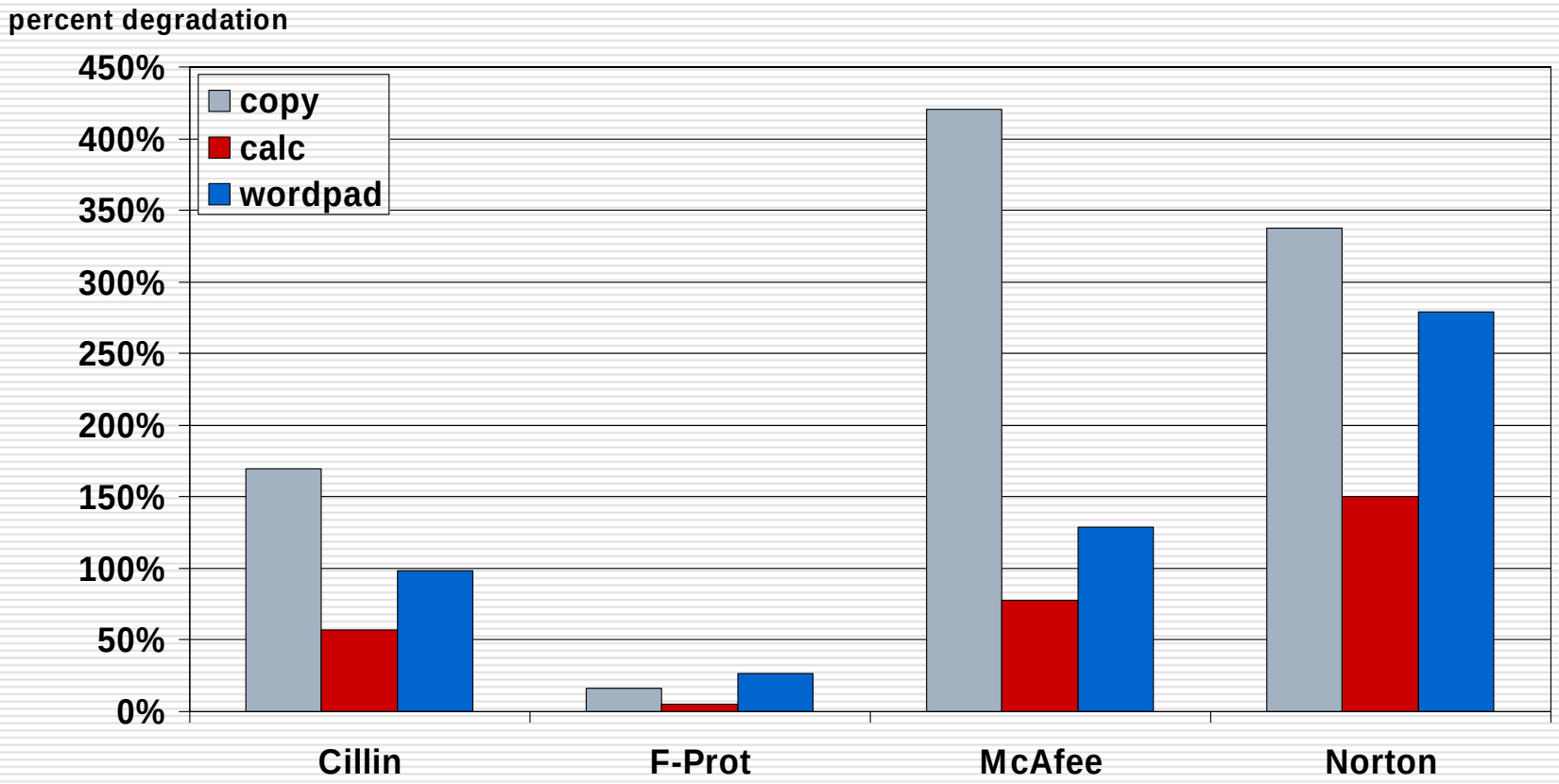
Norton

```
movzx edi, ax
imul edi,dword ptr 0xcfc[edx]
mov ebx,dword ptr 0x10[ebp]
add edi,ecx
cmp ebx,dword ptr [edi]
je 0xf6a13e02
```

- Frequent code exhibits similar structure

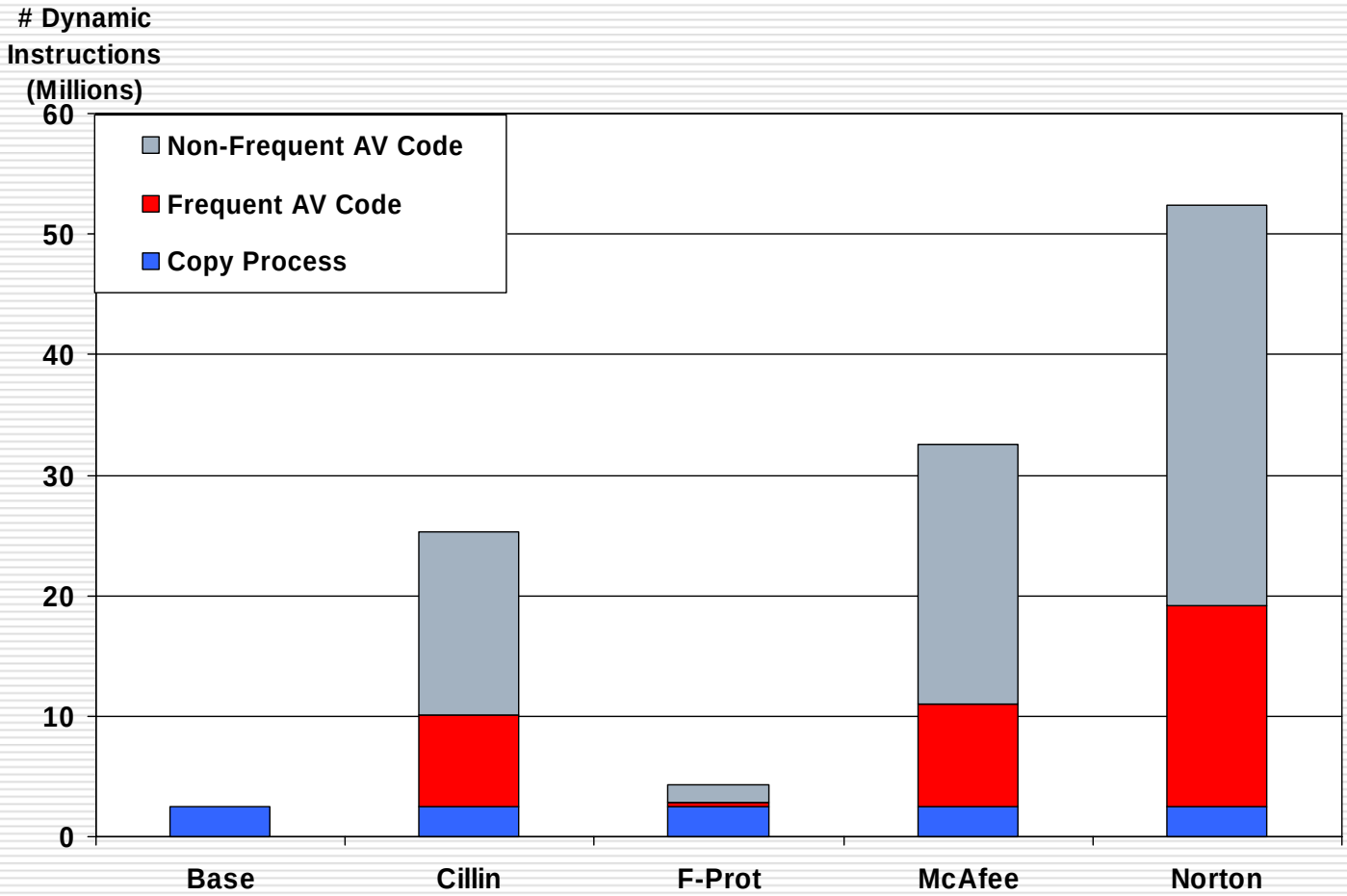
Anti-virus Execution Overhead

Simulated Micro architecture modeling P4 – Relative # cycles
copy (CDROM to disk), executing calc.exe, executing wordpad.exe

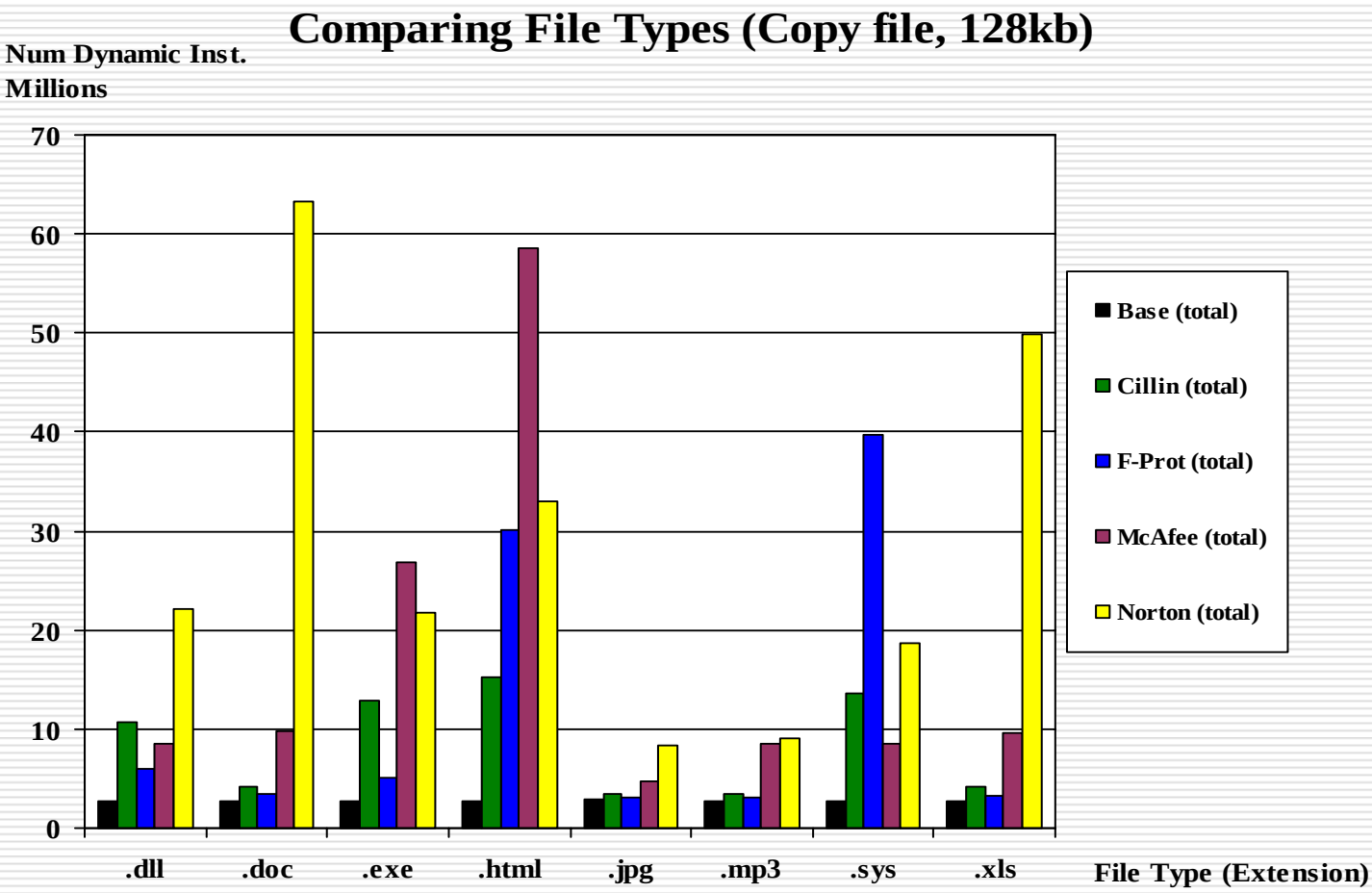


Performance Degradation

copy (CDROM to disk)

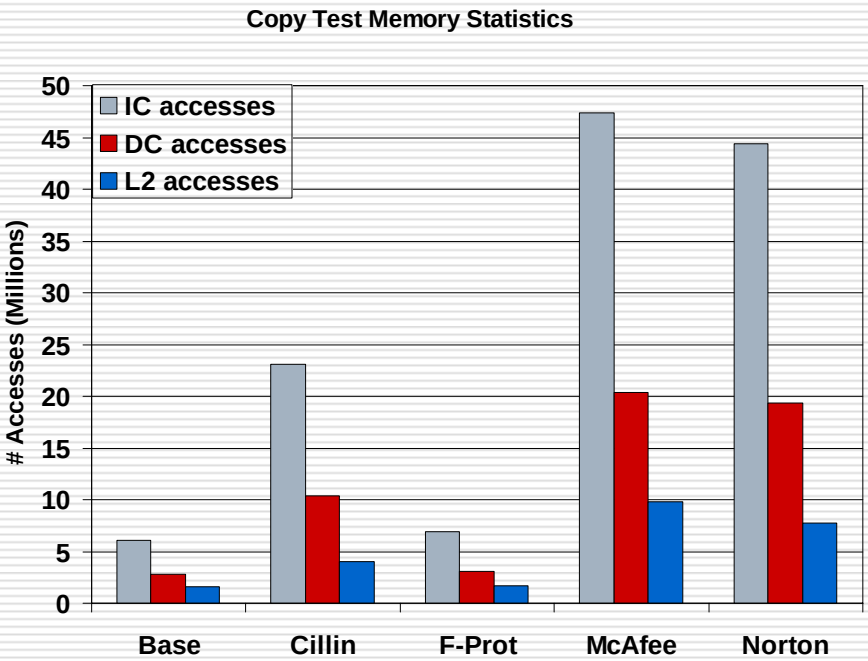


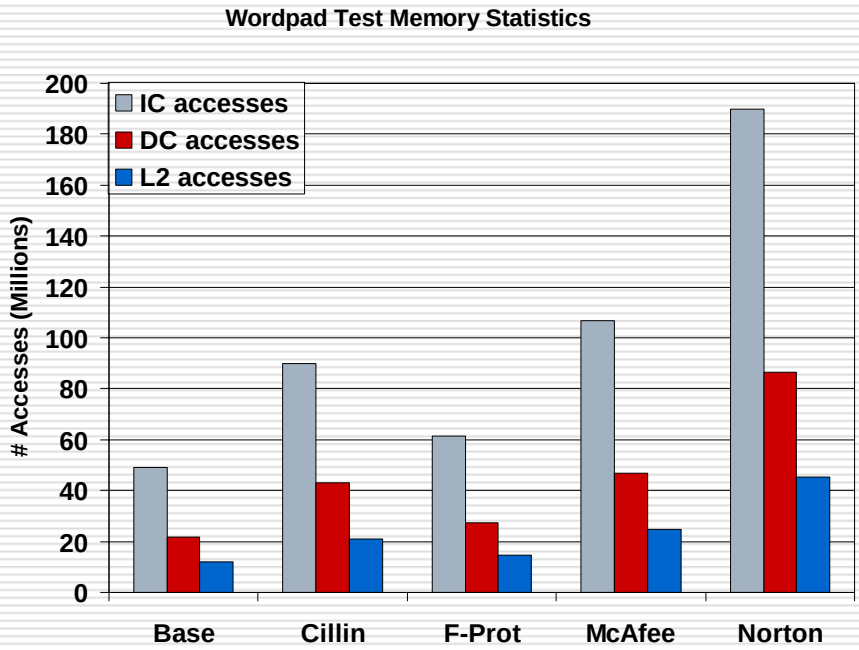
Performance Degradation



Performance Degradation

- Memory Implications – Cache Accesses





Performance Degradation

- Anti-virus workload impact
  - Performance
    - On-Access contributes up to 400% overhead
  - Memory Traffic Overhead
    - IC: ~750%
    - DC:~500%
    - L2: ~350%
- Growing Problem: Input data set grows as more viruses are created

Performance Degradation

Possible Solutions

- Independent “AntiVirus Co-Processor”
  - Processor linked to hard disk to perform real-time scanning duties
  - Alleviate overhead on instruction basis for main CPU
    - Memory overhead still exists
- Memory Controller Enhancement
  - Designed in a filter model
    - Scan instructions and data as they are fetched
  - Transparent to user
  - High maintenance involved
    - Time to update hardware

Performance Degradation

Possible Solutions

- x86 ISA Enhancements
  - Analysis
    - Fuse Instruction Example: mov, cmp, jmp
      - Cillin: 12.77%
      - F-Prot: 21.2%
      - McAfee: 3.7%
      - Norton: 8.8%
- Execute “anti-virus server” scanning mechanism in physical Environment for multiple Virtual Machines

Virtualization Layer Proposal

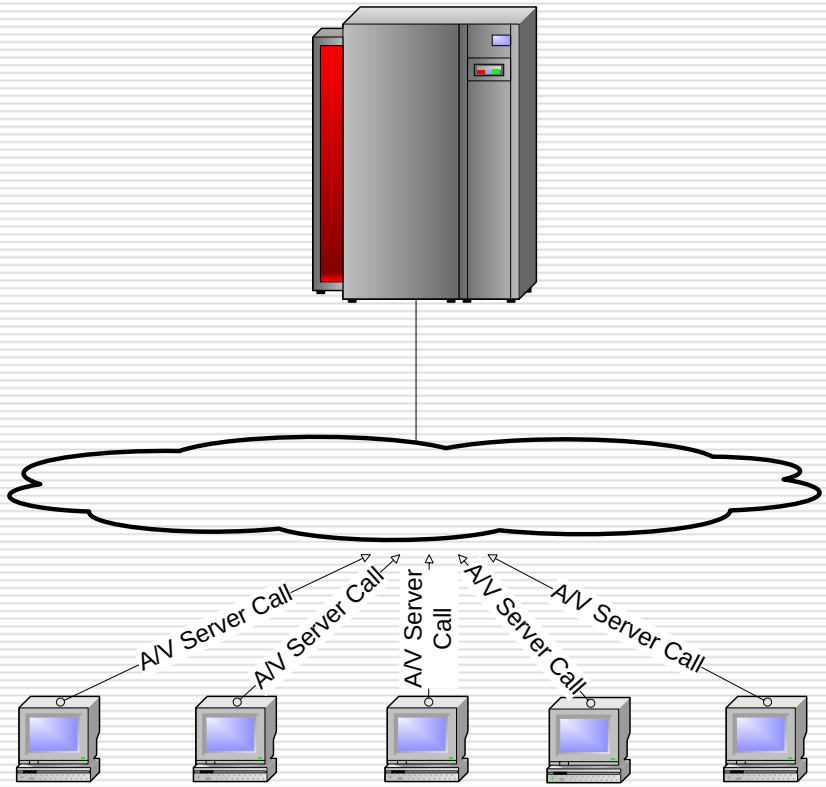
Problem Description

- Multiple virtual Windows clients need anti-virus protection
  - Each client will generate it's own overhead from A/V services
  - This results in more pressure on the physical host

Virtualization Layer Proposal

Proposed Solution

- Logically parallel On-Access A/V Scanning
  - One On-Access process running on the host
  - Called whenever
    - input data passes through the host to a VM (or back)
      - Network
    - A (virtual) system call is made to create or read a file
      - File System



Virtualization Layer Proposal

- One “Host A/V” process would behave similar to a E-Mail A/V Gateway
  - ESX Server performs data filtration testing
  - Implementation involves signature matching
  - Involves only one database of signatures as well as only maintaining a single signature database

Virtualization Layer Proposal

- Implementation
  - On ESX platform, run an internal daemon similar to hostD
  - Communicate with daemon using VMware tools
    - VMware tools has similar behavior to real-time a/v scanning process
    - Send data stream for signature matching
      - System calls to read a file
      - System calls to create a file

Virtualization Layer Proposal

- Communicate with daemon using VMware tools
  - Virtual system uses message passing interface to host A/V daemon
  - If data stream returns positive identification message to VM
    - VMware tools interrupt the system and alerts the user
- Interface to A/V host daemon
  - Virtual Center control panel
    - Allows updates to signature database
    - Statistics: files scanned, viruses found, VM A/V utilization

Conclusions

- Anti-Virus impact
  - CPU and memory overhead
  - Overhead will only increase
- Presents issues for VMs
  - Generates significant load on the host to manage multiple AV filters

Conclusions

- Virtualization Layer Anti-Virus
  - Advantages
    - Reduce the overhead of invoking all A/V functionality on individual machines
      - Still will place overhead of accessing data on machines
    - Consumes less physical system memory
    - Less software to maintain on VM systems

Acknowledgements

- This work was supported by the National Science Foundation award CISE CSA-0310891 and the Institute of Information Assurance

- NUCAR Website
  - http://www.ece.neu.edu/groups/nucar/research/antivirus
- Security Related NUCAR Papers
  - Dong Ye, Micha Moffie and David Kaeli. *A Benchmark Suite for Behavior-Based Security Mechanisms*, Proceedings of the *Workshop on Software Security Assurance Tools, Techniques, and Metrics (SSATTM '05)*
  - Micha Moffie and David Kaeli. *ASM: An Application Security Monitor*, Proceedings of the *Workshop on Binary Instrumentation and Applications (WBIA '05)*
  - Dong Ye and David Kaeli. *A Reliable Return Address Stack: Microarchitectural Features to Defeat Stack Smashing*, Proceedings of the *1st Workshop on Architectural Support for Security and Antivirus (WASSA '04)*
  - Derek Uluski, Micha Moffie and David Kaeli. *Antivirus Workload Execution*, Proceedings of the *1st Workshop on Architectural Support for Security and Antivirus (WASSA '04)*