Analysis of Techniques to Improve Protocol Processing Latency

David Mosberger, Patrick Bridges, Larry L. Peterson, and Sean O'Malley

The University of Arizona

Larry L. Peterson, and Sean O'Malley
David Mosberger, Patrick Bridges,

Protocol Processing Latency Analysis of Techniques to Improve
Latency: Where does it come from?

- Badly structured code.
  - Too much code.
  - Too much code.
  - Execution overheads.
  - No: messages (data) are small.
  - Data touching overheads.
  - Speed of light.

SIGCOMM '96 The University of Arizona
Test Environment

- 10Mbps Ethernet
- TURBOchannel bus
- 100MB/s memory
- 175MHz Alpha

Hardware Platform

- RPC
- TCP/IP

Protocol Stacks

Test Environment

SIGCOMM '96 The University of Arizona
Starting Point

- Instruction count
- Machine idiosyncrasies
- Tiny functions
- Data cache footprint
- Integer division
- Byte load/store
- Into duplication
- Stack switching
- Padding

Injection

Orig Opt

4750
5821
18941
15688
18941
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
How fast is TCP/IP?
Latency Bottlenecks

Not instruction/data translation buffer

Layering overheads

Cache collisions

Instruction-cache gaps

Frequent branching

Suspects

SIGCOMM '96 The University of Arizona
Techniques

- Layering overheads
  - Path-inlining attacks
    - cache collisions
    - i-cache gaps
  - Inlining branching
  - Outlining attacks

Techniques
Outlining

Exception-handling code ± lots of it (up to 50%)

- dilutes instruction-cache
- causes taken branches
- dilutes instruction-cache
- move unlikely code to end of function
- annotate if-statement with branch probability

- remove from fast path
Outlining Example

```c
// Code example from SIGCOMM '96
Outlining Example

// Outlining Example
```

```c
// Outlining Example
```
Cloning

- Make copy of functions on fast path
- Relocate to avoid conflict misses
- Specialize for a particular use (partial)

Alternative layout algorithms
- bipartite layout
- micro-positioning
  (evaluation)

Cloning
Outlining & Cloning Summary

Standard Layout: After Outlining: After Cloning:

- infrequently executed instructions
- frequently executed code
- copy & relocate

After Cloning:

- Function A
- Function B

After Outlining:

- Function A
- Function B
Path-Inlining

Collapse deeply-nested functions

Assume fast path is known

Advantages

Increases context for optimizer

Removes call-overheads

Compile entire function as single unit

Collapse deeply-nested functions

Removes call-overheads

Increases context for optimizer

Advantages
End-to-End Latency

Roundtrip time in ms:

RPC

BAD STD OPT
365.5
399.2
457.1

TCP

BAD STD OPT
310.8
351
498.8

TCP

BAD STD OPT
351
310.8
498.8

SIGCOMM '96 The University of Arizona
Processing Latency

Processing-time per roundtrip in us:

RPC

BAD STD OPT

189.2

247.1

TCP

BAD STD OPT

191

189.2

247.1

155.5

189.2

247.1

141

288.8

155.5

189.2

247.1

141

288.8

141

288.8

155.5

189.2

247.1

141

288.8

155.5

189.2

247.1

141

288.8

SIGCOMM '96 The University of Arizona
Memory System Performance
Essentially identical performance.

TCP

- RPC

85% 15% %

With Outlining

79% 21% %

No Outlining

Outlining Effectiveness

TCP

SIGCOMM '96 The University of Arizona
Conclusions

Instruction cache bandwidth major bottleneck

Cache collisions not particularly bad

Processor/Memory gap still growing, now:
- 80MB/s memory system
- 100Mbps Ethernet
- 300MHz processor

Instructin cache bandwidth major bottleneck
Conclusions

- Need better (automatic) tools
- Requires "path" notion: see Scout OS
- Cloning and path-inlining
- Relatively convenient
- Readily applicable
- Outlining

Conclusions