Contents

Acknowledgements v

Contents ix

List of Figures xiii

List of Tables xv

Publications xvii

1 Introduction 1

1.1 Overview of Operating System Architectures . . . . . . . . . . . . 4

1.1.1 Monolithic Kernels . . . . . . . . . . . . . . . . . . . . . . 4

1.1.2 Microkernels . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.2 New Multicore Hardware . . . . . . . . . . . . . . . . . . . . . . . 7

1.3 Network Stack . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

1.4 NEWTOS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

1.5 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

1.6 Organization of the Dissertation . . . . . . . . . . . . . . . . . . . 13

2 Keep Net Working 17

On a Dependable and Fast Networking Stack 17

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

2.2 Reliability, performance and multicore . . . . . . . . . . . . . . . . 19

2.3 Rethinking the Internals . . . . . . . . . . . . . . . . . . . . . . . . 21

2.3.1 IPC: What’s the kernel got to do with it? . . . . . . . . . . . . 21

2.3.2 Asynchrony for Performance and Reliability . . . . . . . . . . 22
6.3.8 Multi-component Network Stack .......................... 116
6.4 Architectural Support ........................................ 116
6.5 Evaluation ..................................................... 118
  6.5.1 Scalability on a 12-core AMD ......................... 118
  6.5.2 Scalability on a 8-core Xeon ......................... 120
  6.5.3 Impact of Different Configurations ................. 123
  6.5.4 Reliability ............................................. 124
6.6 Conclusion ................................................... 125

7 Summary and Conclusions .................................. 127

References ....................................................... 133

Summary .......................................................... 147

Samenvatting .................................................... 151
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Conceptual overview of NEWTOS. Each box represents a core. Application (APP) cores are timeshared.</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Decomposition and isolation in multiserver systems</td>
<td>28</td>
</tr>
<tr>
<td>2.3</td>
<td>Asynchrony in the network stack</td>
<td>30</td>
</tr>
<tr>
<td>2.4</td>
<td>IP crash</td>
<td>36</td>
</tr>
<tr>
<td>2.5</td>
<td>Packet filter crash</td>
<td>37</td>
</tr>
<tr>
<td>3.1</td>
<td>Design of the NEWTOS network stack</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Test configurations - large squares represent the quad-core chips, small squares individual cores and ‘&amp;’ separates processes running on different hyper-treads.</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>Throughput compared to the combined performance of the resources in use — the best combinations</td>
<td>54</td>
</tr>
<tr>
<td>3.4</td>
<td>Configuration #1 – CPU utilization of each core throttled to % of 1600 MHz.</td>
<td>55</td>
</tr>
<tr>
<td>3.5</td>
<td>Configuration #1 – CPU utilization normalized to cores at 1600 MHz unthrottled.</td>
<td>55</td>
</tr>
<tr>
<td>3.6</td>
<td>Configuration #2 (HT) – CPU utilization of each core throttled to % of 1600 MHz.</td>
<td>56</td>
</tr>
<tr>
<td>3.7</td>
<td>Configuration #2 (HT) – CPU utilization of each core normalized to 1600 MHz.</td>
<td>56</td>
</tr>
<tr>
<td>3.8</td>
<td>Comparison of configuration #1 (no HT) and #2 (HT). Lines represent bitrate, bars represent CPU utilization normalized to 3 cores at 1600 MHz</td>
<td>57</td>
</tr>
<tr>
<td>3.9</td>
<td>Sleep overhead in different situations. The thick line represents transition between execution in user space and kernel in time. Arrows mark job arrivals, loops denote execution of the main loop and spirals denote polling.</td>
<td>58</td>
</tr>
</tbody>
</table>
4.1  NEWTOS- network stack and inputs of the scheduler .......................... 66
4.2  NEWTOS vs Linux performance. Continuously requesting 10 times a
10 B file using various numbers of simultaneous persistent connections. 67
4.3  Results for tests requesting continuously a file of a size between 10 bytes
and 10 Megabytes using 8 simultaneous connections. The dashed line in
(a) and (g) connects points where all three components run with the same
frequency. 10 MB case shows bitrate instead of request rate. Legend is
only in (f). .......................................................... 70

5.1  Net stack configurations: (a) Monolithic systems (Linux, BSD, Win-
dows, etc.) (b) FlexSC / IsoStack (c) Multiserver system (MINIX 3,
QNX). ................................................................. 79
5.2  Exposed socket buffers – no need to cross the protection boundary (dashed)
to access the socket buffers. ........................................ 83
5.3  A socket buffer – ring queue and data buffer. White and shaded present
free and used / allocated space. ................................. 84
5.4  Architecture of the network stack ............................................. 89
5.5  1 request for a 20 B file per connection .................................... 94
5.6  10 requests-responses per connection, 20 B file ......................... 95
5.7  1 request and a 11 kB response per connection ............................ 96
5.8  10 requests for an 11 kB file .................................................. 97
5.9  1 request for a 512 kB file per connection ................................. 99
5.10 Memcached - test duration for different size of stored values. Test uses
128 clients. Smaller is better. ........................................... 99

6.1  NEAT: a 4-replica example. ....................................................... 108
6.2  A dedicated path for packets of each connection. ........................... 109
6.3  Interrupts and system load distribution of a highly loaded memcached
server—a hypothetical example. ....................................... 111
6.4  A replica of the multi-component network stack ............................ 115
6.5  12-core AMD - The best-performing setups using all 12 cores .......... 119
6.6  AMD - Scaling lighttpd and the network stack ............................. 119
6.7  NEAT scales the required number of cores ................................. 120
6.8  Colocated multi-component configuration of NEAT ........................ 121
6.9  Xeon - Scaling the multi-component network stack ........................ 121
6.10 Xeon - The best-performing configuration of the single-component stack,
fully exploiting hyper-threading (HT). ................................... 122
6.11 Xeon - Scaling the single-component network stack ........................ 122
6.12 12-core AMD - Comparing performance of different network stack con-
figurations stressed by the same workload. ............................ 123
6.13 Expected fraction of state preserved after a failure vs. max throughput
across different network stack setups. ................................ 124
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Complexity of recovering a component</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>Peak performance of outgoing TCP in various setups</td>
<td>34</td>
</tr>
<tr>
<td>2.3</td>
<td>Distributions of crashes in the stack</td>
<td>35</td>
</tr>
<tr>
<td>2.4</td>
<td>Consequences of a crashes</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of Core i7, Knights Ferry and Xeon Phi</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>Given the known transistor counts shown in (a) and a 22nm production process, we can roughly project the options for different configurations that merge Core i7 and Xeon Phi cores (b)</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance loss versus potentially saved power</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>Bitrate vs. CPU clock speed on a single core</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>Network Traffic Indicators</td>
<td>68</td>
</tr>
<tr>
<td>5.1</td>
<td>Cycles to complete a nonblocking <code>recvfrom()</code></td>
<td>81</td>
</tr>
<tr>
<td>5.2</td>
<td>Pipelined 10×11 kB test – percentage of calls handled in user space, fraction of selects that block, ratio of selects to all calls, accepts and reads that would block</td>
<td>98</td>
</tr>
<tr>
<td>6.1</td>
<td>lighttpd I-cache miss rate on Linux and on a dedicated core in NEAT, with or without using system calls.</td>
<td>112</td>
</tr>
<tr>
<td>6.2</td>
<td>10G driver - CPU usage breakdown.</td>
<td>124</td>
</tr>
</tbody>
</table>