Scalable Cyber-Security for Terabit Cloud Computing

2012 IEEE High Performance Extreme Computing

Jordi Ros-Giralt / giralt@reservoir.com

Reservoir Labs
• To be able to do packet analysis at very high-speed rates.
• To be able to intelligently move packets from node to node at very high-speed rates.

Two fundamental problems: packet analysis and packet forwarding

Feature Needed (at very high speed rates): Application:

Packet analysis → Cyber security
Packet forwarding → Cloud computing / load balancing
PAAS model – the consumer has control over the deployed cyber-security applications and their configuration and it provides support for any of the four deployment models (private cloud, community cloud, public cloud, hybrid cloud)
Packet Forwarding and Packet Analysis: Two Faces of the Same Coin (Ex Ante and Ex Post Configurations)

- Optimizing for \{\text{Speed} + \text{Analysis}\} \neq \text{Optimizing for Speed} + \text{Optimizing for Analysis}
- Optimality is achieved with the joint problem

- aCSC cells -- a network configuration to address the jointly optimized solution:

(a) \hspace{1cm} (b) \hspace{1cm} (c)

- traffic
- cyber data

1: receive
2: analyze
3: process cyber data
4a: forward
4b: drop

1: receive
2a: forward
2b: drop
Mcore: Mapping Architecture onto a Manycore

Diagram showing the mapping architecture onto a manycore, with TED queuing and feedback paths between components labeled as aCSC1 and aCSC2, connected by mPIPE.
<table>
<thead>
<tr>
<th>Feature:</th>
<th>Utility:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mPIPE: Programmable packet classification and switching engine with up to 80G bps and 120M packets-per-second of throughput</td>
<td>- High-speed packet switching</td>
</tr>
<tr>
<td>TED Queuing (Tail Early Dropping): Intelligent congestion-avoidance packet dropping policy.</td>
<td>- No kernel involvement</td>
</tr>
<tr>
<td>LF - Data Structure: Lock-free data structure with low probability of false negative to convey feedback from A to F</td>
<td>- Intelligent packet dropping</td>
</tr>
<tr>
<td></td>
<td>- Designed to make the system operate out of cache</td>
</tr>
<tr>
<td></td>
<td>- Zero locks everywhere</td>
</tr>
<tr>
<td></td>
<td>- Minimize memory contention</td>
</tr>
</tbody>
</table>
• Suppose that the system is so stressed that we need to drop a packet. In the context of CSC, which packet should we drop?
• Approach: leverage the flows’ heavy tails

\[ b_n: n\text{-th bit received from a flow} \]
\[ I(b_n): \text{degree of cyber security information carried by bit } n\text{-th} \]
TED Queuing designed to (1) exploit heavy tails and (2) to find an optimal trade-off in hierarchical memory architectures.
TED Queuing

- TED queuing principle:

![Diagram showing TED queuing principle]

Lemma 1: Let $c_{in}$ be the amount of cyber data per unit of time received by the node and let $\pi_c$ be the amount of cyber data per unit of time that it can actually process:

- If $\pi_c < c_{in}$, there exists a value $\lambda_{ted}^h$ such that for any $\lambda_{ted} > \lambda_{ted}^h$, the analyzers in the node will be accessing packets from memory (cache miss).
- For any value of $\pi_c$, there exists a value $\lambda_{ted}^l$ such that for any $\lambda_{ted} < \lambda_{ted}^l$, the analyzers in the node will be accessing packets from cache (cache hit).
TED Queuing

- TED queuing algorithm:

  Constants: $\Delta_{ted}, t_{ted}, \lambda_{ted}^i$.
  
  **Step 1.** Start with $\lambda_{ted} = \lambda_{ted}^i$, for an arbitrary $\lambda_{ted}^i$.
  
  **Step 2.** If the cell is operating in memory regime, then keep reducing $\lambda_{ted}$ by a value $\Delta_{ted} > 0$ until the cell starts to operate in cache regime.
  
  **Step 3.** Wait a period of time $t_{ted}$ and then increase $\lambda_{ted}$ by $\Delta_{ted}$. After that, return to step 2.

![Graphical representation of TED queuing algorithm](image)
Data structure designed to satisfy the following specifications:
- It must be able to keep at least one flag for each element stored.
- It can be concurrently written and read by multiple writers and multiple readers but it does not require locks to preserve the correctness of its elements.
- It can tolerate a low probability of false negatives but not false positives.

Algorithm:

- Initial state: $T[e] = \text{NULL}$ for all $e$ such that $0 \leq e < N$;
- Writer (analyzer) algorithm:
  - Upon detecting that $c$ needs to be dropped, do:
    $$T[h(id(c)) \mod N] = h(id(c));$$
- Reader (forwarder) algorithm:
  - Upon receiving a packet from connection $c$, do:
    - If $T[h(id(c)) \mod N] = h(id(c))$
      - Drop the packet;
    - Otherwise
      - Forward the packet;
Lemma 1: LF^− state correctness. An element in a data structure constructed using the LF^− algorithm is in a positive state with probability \( p_t \), in a false negative state with probability \( p_{fn} \), and in a false positive state with probability \( p_{fp} \), where \( p_{fp} \ll p_{fn} \ll p_t \) and \( p_t \approx 1 \).

Proof. [Working out the math—see HPEC12 paper]

\[
p_t \approx \frac{1}{(1 + \frac{1 - p_{t,f}}{p_{fn,t}})}
\]

\[
p_{fn} \approx \frac{p_{t,fn}}{(1 - p_{fn,fn})}
\]

\[
p_{fp} \approx \frac{p_{t,fp}}{(1 - p_{fp,fp})}
\]
Tilera Gx

aCSC₁

broₐ broₐ broₐ

(...)

broᵢ LF-table

(...)

aCSC₂

broₐ broₐ broₐ

feedback

feedback

mPIPE

TED queuing

Reservoir Labs
Tilera Gx (TILEExtreme)

Manycore platform:
- 36 tiles per processor x 4
- 4 x 10Gbe per processor
- Roadmap for 100 tiles per processor
- mPIPE: I/O acceleration
- MiCA: crypto acceleration
Tilera Inter-Processor Communication

80 port PCIe switch

aCSC_n
bro_a
bro_a
bro_a

feedback

bro_f
LF-table

mPIPE

Reservoir Labs
Tilera Inter-Processor Communication

80 port PCIe switch

PCIe (32Gbps)

TRIO

mPPipe

Reservoir Labs
mcore-def.bro

type mcore_node_type: enum { NULL, FORWARDER, ANALYZER, FORWARDER_ANALYZER };
type mcore_node_id: count;

const mcore_interfaces_input: set[string] = {
} &redef;

const mcore_interfaces_output: set[string] = {
} &redef;

const mcore_do_shunting = F &redef;
const mcore_ted_flow_threshold = 2000 &redef;  # in number of bytes within a flow
const mcore_ted_queue_threshold = 50  &redef;  # in % of queue utilization from 0 to 99
const mcore_force = F &redef;

const mcore_enabled = F &redef;
const mcore_is_manager = F &redef;

const mcore_node_map: table[mcore_node_id] of mcore_node_type = {
} &redef;
mcore-x-y-z.bro

redef mcore_node_map = {
    [0] = FORWARDER,
    [1] = FORWARDER,
    (…)  
    [x-1] = FORWARDER,
    [x]  = ANALYZER,
    [x+1] = ANALYZER,
    (…)  
    [x+y-1] = ANALYZER,
    [x+y]  = FORWARDER_ANALYZER,
    [x+y+1] = FORWARDER_ANALYZER,
    (…)  
    [x+y+z-1] = FORWARDER_ANALYZER,
};

arch/tilera/mcore.h

void mcore_get_ring_writer(void);
void mcore_get_ring_reader(void);
int mcore_conf_rings(int num_analyzers, int num_forwarders);
int mcore_init_rings(int NumAnalyzers);
int mcore_link_init(int *cpu_ranks,
    int num_forwarders,
    int num_rings,
    char* if_input,
    unsigned char mac_output[][ADDR_LEN],
    char if_output[][MAX_STRLEN],
    int num_output_if,
    int num_input_if);
void * mcore_alloc_shmem(unsigned int size);
void mcore_free_shmem(void * mem, unsigned int size);
## Performance (Tile-GX)

### Analytics: Bro's default set + http_track.bro

<table>
<thead>
<tr>
<th># Forwarders</th>
<th># Analyzers</th>
<th>Selective Capacity (events/sec)</th>
<th>Forwarding throughput (Gbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Per Tile</td>
<td>Total</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>98.920041</td>
<td>98.920041</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>167.699023</td>
<td>335.398046</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>194.169252</td>
<td>582.507756</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>152.70955</td>
<td>610.8382</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>122.405928</td>
<td>612.02964</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>93.275352</td>
<td>559.652112</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>66.541857</td>
<td>465.792999</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>47.263248</td>
<td>378.105984</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>164.556612</td>
<td>329.113224</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>148.848772</td>
<td>595.395088</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>91.04497</td>
<td>728.355976</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>168.595005</td>
<td>674.38002</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>94.797063</td>
<td>758.376504</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>96.312077</td>
<td>770.496616</td>
</tr>
</tbody>
</table>

### Diagram:

- **1 Forwarder**
- **2 Forwarders**
- **4 Forwarders**
- **8 Forwarders**

The diagram illustrates the relationship between the number of analyzers and selective capacity, showing how the capacity increases as the number of analyzers increases, with different configurations achieving varying levels of performance.
• Live demo at the SCinet Research Sandbox this coming November → target: 4 x Tile Gx-36 from 10 to 80 Gbps.
• Tilera Gx roadmap toward 100 core processors. Leverage this roadmap and other development in the manycore space toward scaling to terabit computing.
• Looking for testbeds that may be interested in testing Mcore technology.
References

• S. Campbell, J. Lee, “Intrusion Detection at 100G”, The International Conference for High Performance Computing, Networking, Storage, and Analysis, November 14, 2011
• J. Ros-Giralt, P. Szilagyi, “A Lockless Hash Table With Low False Negatives,” mathematical note, Reservoir Labs, April 2012.
• Carl Ramey, “TILE-Gx100 ManyCore Processor: Acceleration Interfaces and Architecture,” Tilera Corporation, August 2011.