

TCP/IP in hardware using SME

Mark Jan Jacobi & Jan Meznik

KU

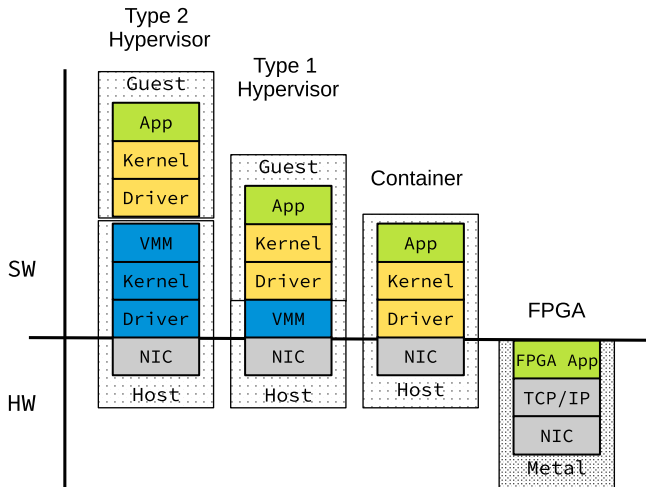
September 16, 2019

Table of Contents

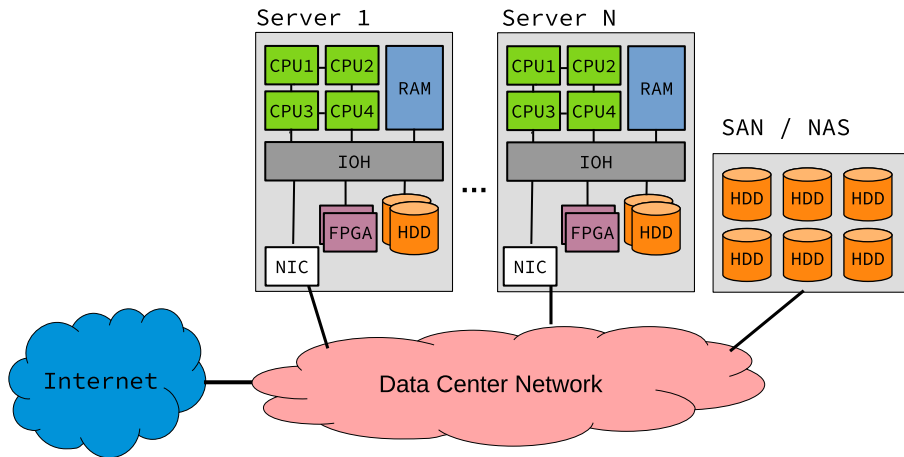
- 1 Introduction
- 2 Implementation
- 3 Evaluation
- 4 Discussion
- 5 Conclusion
- 6 Future Work
- 7 Questions

Background and Motivation

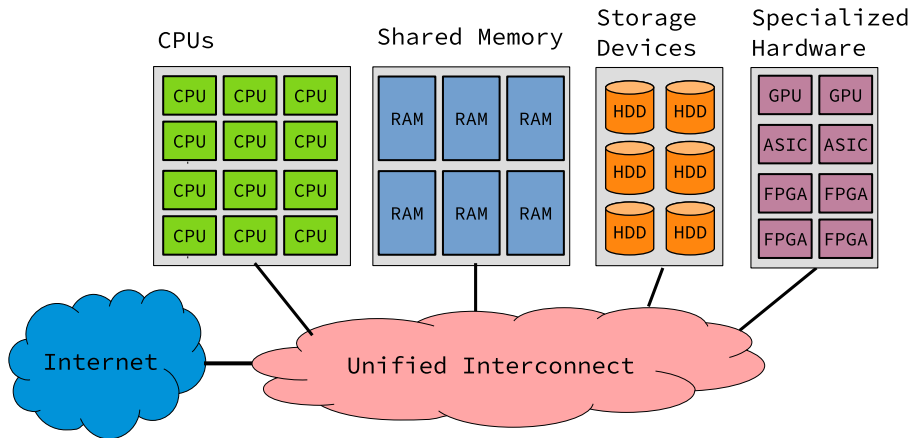
FPGAs are making their way into data centers to boost the computing power and the overall power efficiency.



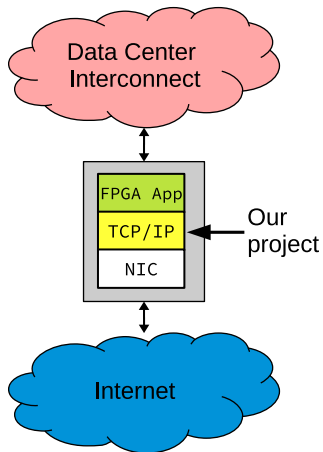
A conventional data center architecture



Proposed, disaggregated data center architecture



FPGA usage



The Internet

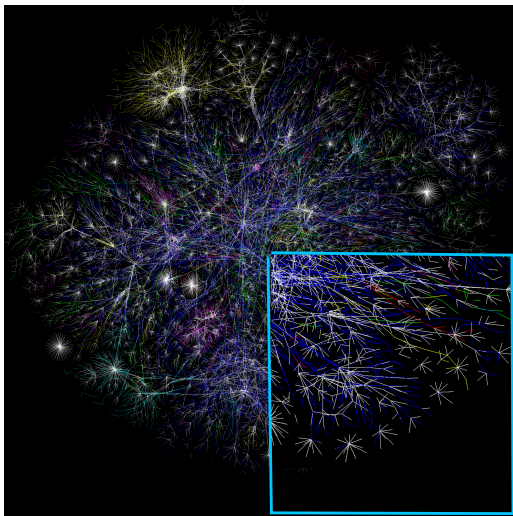
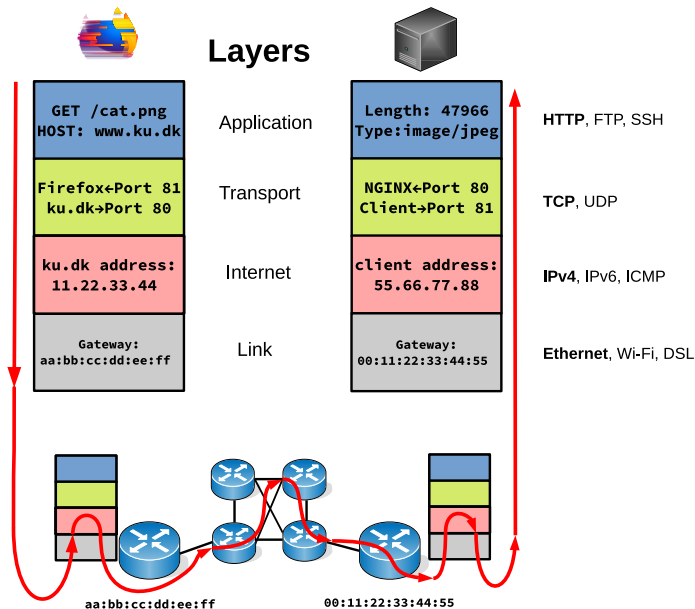


Figure: Map of the 30% of accessible the endpoints on the Internet

The Internet Protocol Suite – A scenario



Design with the 4 layers in mind

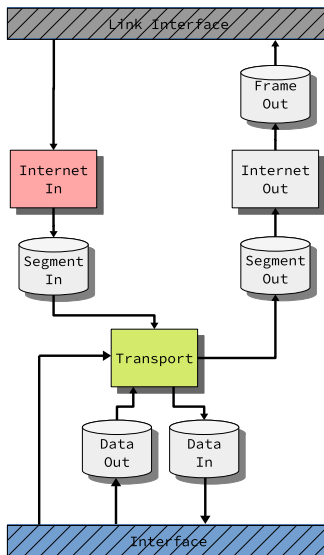


Table of Contents

- 1 Introduction
- 2 Implementation**
- 3 Evaluation
- 4 Discussion
- 5 Conclusion
- 6 Future Work
- 7 Questions

- Processes
 - State machines
- Buffers
 - Memory segments
 - Dictionary
- Interface signal control
 - Buffer-Producer
 - Compute-Producer
- Interface control
 - Usage
 - Limitations

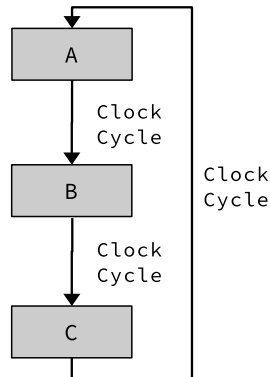
Implementation

Processes

State machines

```
1 public class SomeProcess :  
2   ↳ StateProcess  
3 {  
4     private override async  
5     ↳ Task OnTickAsync()  
6     {  
7       a();  
8       await ClockAsync();  
9       b();  
10      await ClockAsync();  
11      c();  
12      await ClockAsync();  
13    }  
14  }
```

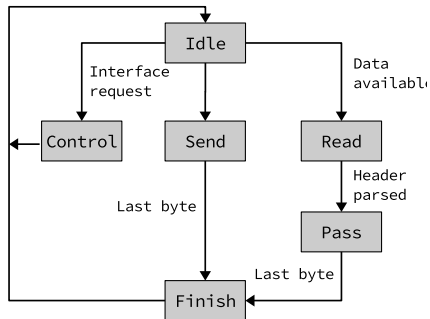
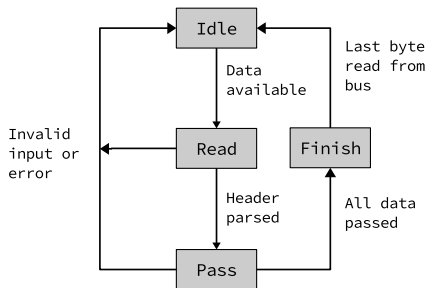
```
1 public class SomeProcess :  
2   ↳ SimpleProcess  
3 {  
4   // Initial state  
5   state = A;  
6  
7   protected override void  
8   ↳ OnTick()  
9   {  
10    switch(state) {  
11      case A:  
12        a();  
13        state = B;  
14      case B:  
15        b();  
16        state = C;  
17      case C:  
18        c();  
19        state = A;  
20    }  
21  }
```



Implementation

Processes

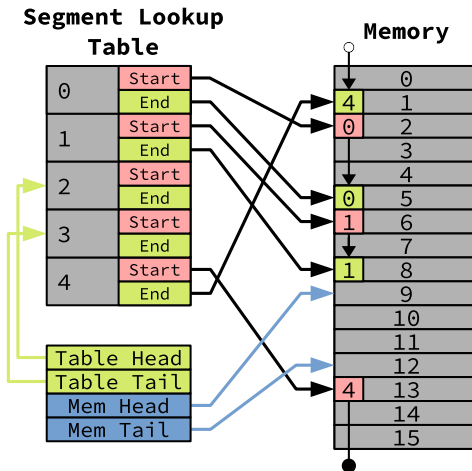
Examples



Implementation

Buffers

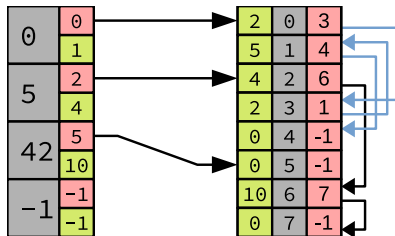
Memory segments



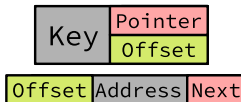
Implementation

Buffers

Memory dictionary



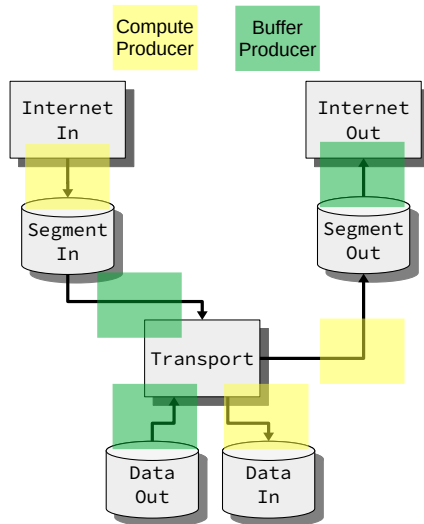
Value Legend



Implementation

Interface signal protocol

Identifying the scenarios

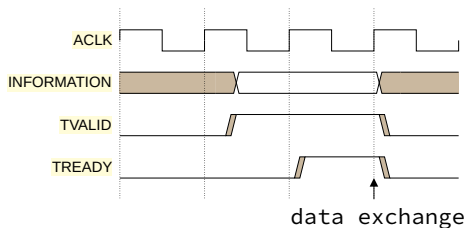


Implementation

Interface signal protocol

Inspired by AXI4

- Single clock offset when sending data.
- Indicate end of stream with `bytes_left`.

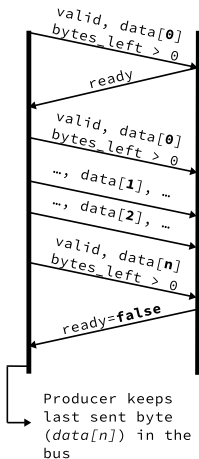


Implementation

Interface signal protocol

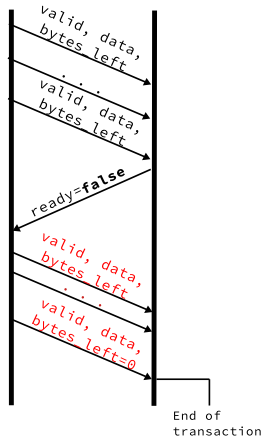
Buffer-Producer (BP)

Producer Consumer



Compute-Producer (CP)

Producer Consumer



Implementation

Interface protocol

The interface structures

```
1  enum InterfaceFunction : byte
2  {
3      INVALID = 0,
4      // BIND = 1,
5      LISTEN = 2,
6      CONNECT = 3,
7      ACCEPT = 4,
8      CLOSE = 7,
9      // ...
10     OPEN = 255,
11 }
12
13 struct InterfaceData
14 {
15     public int socket;
16     public uint ip;
17     public byte protocol;
18     public ushort port;
19 }
```

```
1  interface InterfaceBus : IBus
2  {
3      bool valid;
4      byte interface_function;
5      InterfaceData request;
6  }
7
8  interface InterfaceControlBus : IBus
9  {
10     bool valid;
11
12     byte exit_status;
13     byte interface_function;
14     InterfaceData request;
15     InterfaceData response;
16 }
```

Implementation

Interface protocol

Limitations

- One request at a time.
- Arbitrary delay between request and response.

Table of Contents

- 1 Introduction
- 2 Implementation
- 3 Evaluation**
- 4 Discussion
- 5 Conclusion
- 6 Future Work
- 7 Questions

- Setup
 - Graph file simulator
- Test
- Validation
 - Latency
 - Outgoing packet validation
 - Internet Protocol Suite compliancy as per RFC 1122

Graph file simulation

- Full input - output
- Does not take latency between packets into account
- Simplifies test cases

Evaluation

Setup

Graph simulation node types



Data In



Data Out



Send



Receive



Command



Wait

State and Color

Description

Waiting

Vertex is not in use.

Ready

Vertex is ready
for activation.

Active

Vertex is active.
Simulator is
gathering data.

Inactive

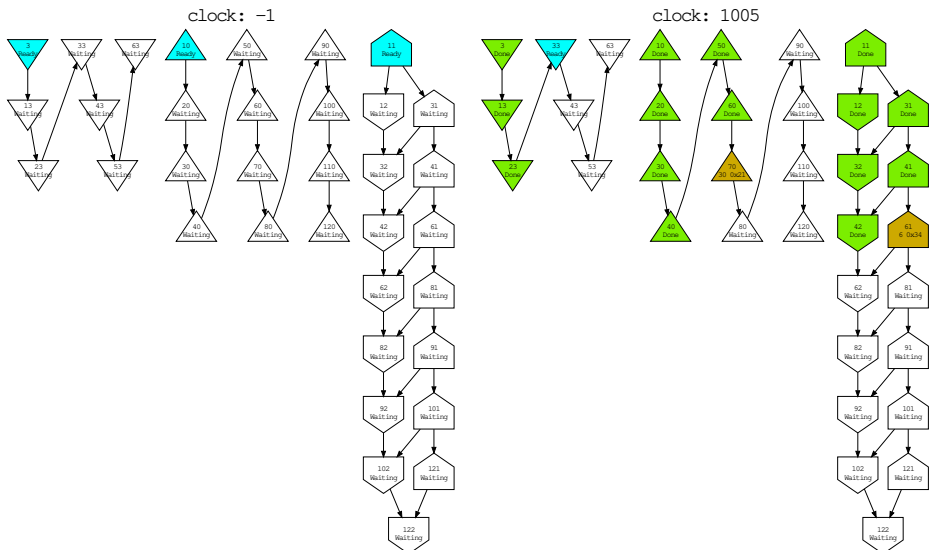
Vertex is inactive.
Simulator is not
gathering data.

Done

Vertex is done
and validated.

Evaluation

Setup



Senario

- Real life scenario
- Test at high workloads
- Remove garbage
- Respond to packet
- Differ between concurrent connections

The test

- 17283 packets in total
- Two "sessions"
- 640*2 UDP packets that needs a response
- 640 well formed UDP packets with no session (discard)
- Rest of data is "background noise" (TCP packets with state, data, etc)
- Total data sent through: 1832958 bytes
- 1.83 Million clocks used

Latency calculations:

n_D : The number of bytes in the data part of the protocol. This excludes both headers from transport and internet.

n_I : The internet header size.

n_T : The transport header size.

n : The total packet size.

From packet to user

$$6 + n_I + 2n_T + 3n_D$$

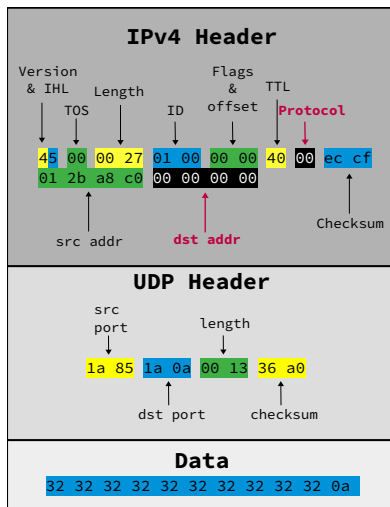
From user to packet

$$8 + 2n_I + 3n_T + 4n_D$$

Evaluation

Validation

Outgoing packet validation:



Internet Protocol Suite compliancy as per RFC 1122

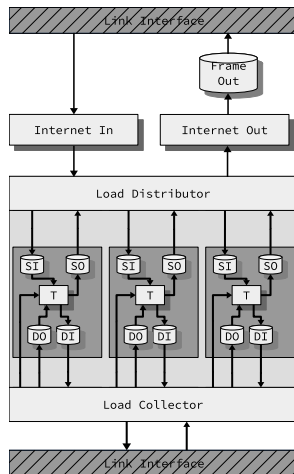
Table of Contents

- 1 Introduction
- 2 Implementation
- 3 Evaluation
- 4 Discussion**
- 5 Conclusion
- 6 Future Work
- 7 Questions

Improving the performance:

Estimated performance:

$$1 \text{ Byte} * 10 \text{ MHz} = 80 \text{ Mbps}$$



Usability

SOMETHING

Using C#

State modelling
Simulation
Concurrency

Table of Contents

- 1 Introduction
- 2 Implementation
- 3 Evaluation
- 4 Discussion
- 5 Conclusion**
- 6 Future Work
- 7 Questions

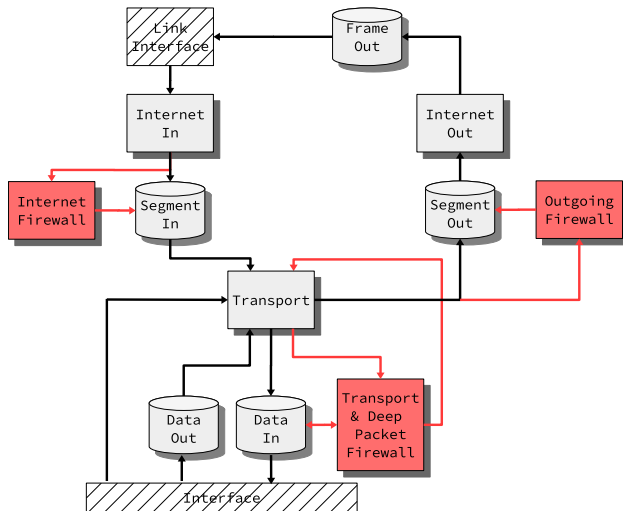
- Lot of trial and error to find the optimal design in the beginning
- In 10 mio. simulated clock cycles, 17283 packets were handled, 1280 of which were correctly received by the Application layer, and then sent out again
- Even with a few flaws, SME is a great framework for hardware modelling

Table of Contents

- 1 Introduction
- 2 Implementation
- 3 Evaluation
- 4 Discussion
- 5 Conclusion
- 6 Future Work**
- 7 Questions

Future Work

Firewall



Future Work

TCP

How to implement TCP

Table of Contents

- 1 Introduction
- 2 Implementation
- 3 Evaluation
- 4 Discussion
- 5 Conclusion
- 6 Future Work
- 7 Questions**

? Some random citation so it does not
complain[Andrew S Tanenbaum(2013)]



Todd Austin Andrew S Tanenbaum.

Structured computer organization.

Pearson, Boston, 2013.

ISBN 978-0-273-76924-8.

end

end