# Neuromorphic Hardware in Outer Space: Software Defined Networking Executed on an In-Orbit Loihi Spiking Processor


Tarek Taha
Md. Nayim Rahman
Chris Yakopcic

Brisk Computing, LLC
*ttaha@ieee.org*

Ricardo Lent

University of Houston
*rlent@central.uh.edu*


June 20, 2023

# Overview

- Cognitive network,
- Software-defined Networking (SDN),
- Network architecture and implementation,
- Neuromorphic processor as cognitive agent (Intel’s Loihi)
- Testbed experimental results and energy calculation
- Implementation in outer space

# Cognitive Network

BRISK COMPUTING

- In cognitive networking, networking issues are dealt with autonomously by observing and collecting information from the environment and making appropriate decisions to achieve a higher level of automation.
- Cognitive network enabled devices adapt to changes in the network environment or user demand without human intervention.
- Cognitive networking improves the performance and efficiency of the network.
- Applications:
  - Space exploration missions where the data transmissions occur over long unreliable channels
  - selecting new network paths, or managing the allocation and deallocation of computing resources

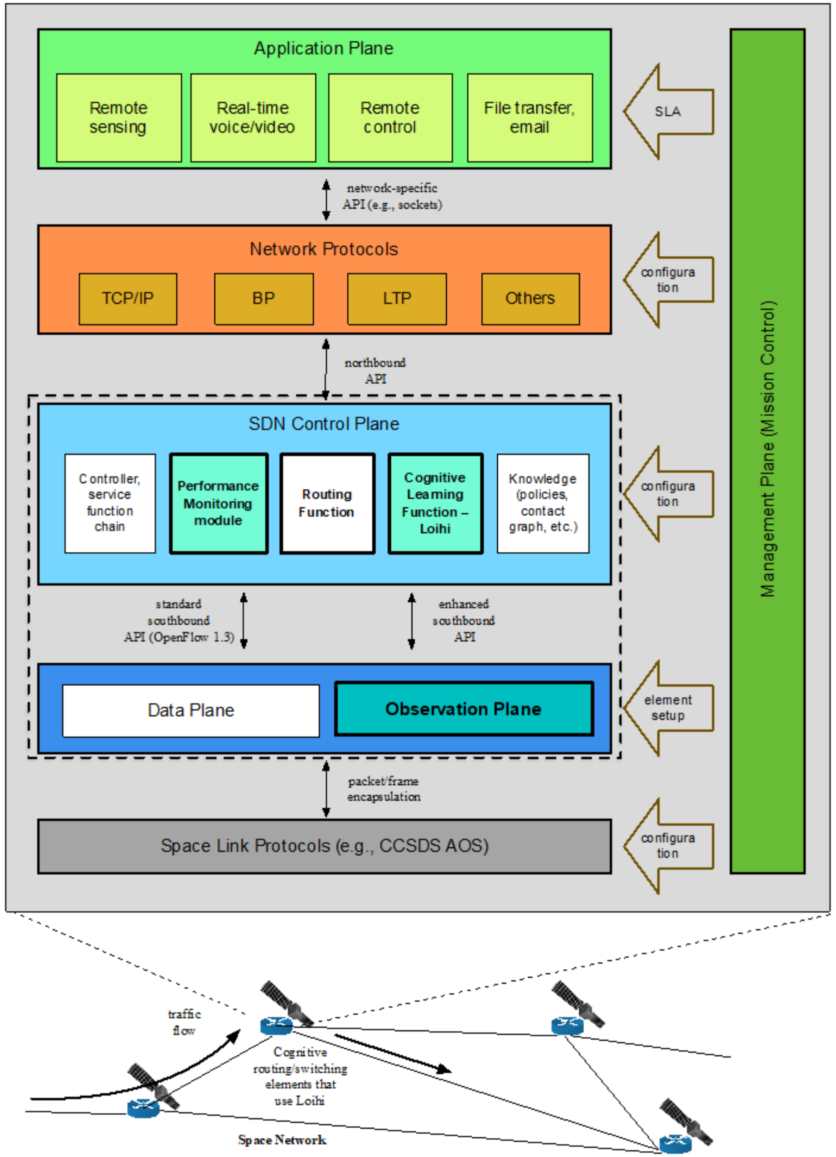
# Software-Defined Networking (SDN)

- SDN uses software-based controller or application programming interfaces to control the communication of network hardware infrastructure and data traffic.
- Benefits:
    - More flexibility and control on the network operation.
    - Customizability of the network operation.
    - Robust security.

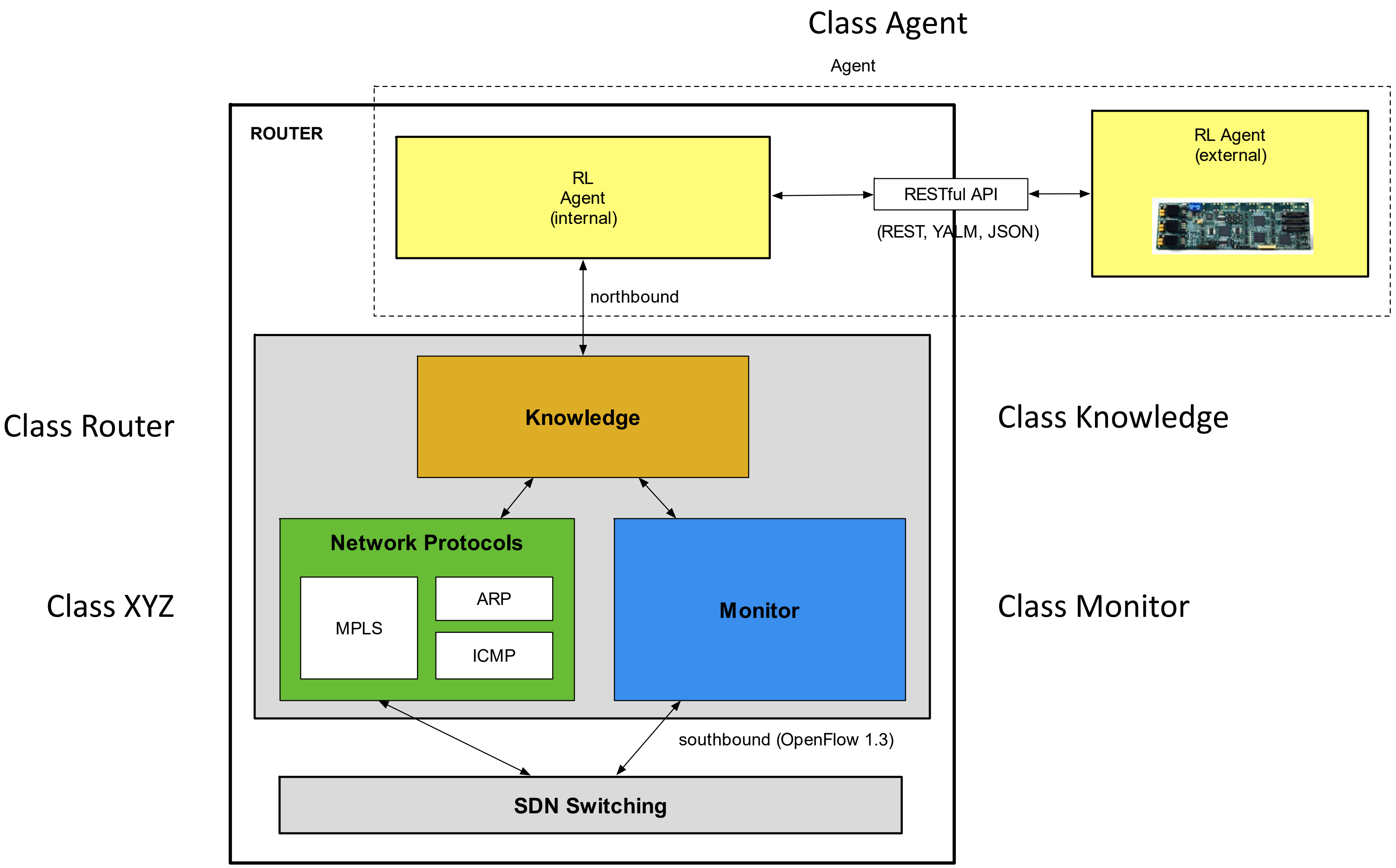
# Cognitive Network Architecture

BRISK COMPUTING

- ❑Designed network architecture:
  - ➢Based on SDN
  - ➢Dynamic environment
  - ➢Supports dynamic routing management
- ❑Continually maps active flows to paths, which change according to an assigned goal and the state of the links



# Software Architecture



# Design Principles

BRISK COMPUTING

- ❑ Architecture defined around the *Ryu* controller and MPLS (Multiprotocol label switching)
- ❑ Routing is handled by a virtual network function (VNF)
- ❑ Flows are defined as a 5-tuple (IP src, IP src_port, IP dst, IP dst_port, protocol)
- ❑ The MPLS network core consists of LER and LSR nodes
  - Label-based, no IP
  - Ingress LER maps a new flow to the ingress label
  - Labels are distributed along the selected path
  - Egress LER removes the label and forwards out the IP packets
- ❑ Expected network graph and IP mappings are given:
  - At boot time (YALM file)
  - Dynamically updated through a REST interface
  - Additional information is acquired dynamically
  - Become part of the *knowledge*


```
#routing: qlearning
routing: random
#routing: shortestpath
#routing: static

graph:
  s22:
    s23: 3
    s24: 1
    s25: 2
  s23:
    s22: 2
    s21: 3
    s26: 1
    s24: 4
  s21:
    s23: 3
    s24: 1
#    s25: 2
  s24:
    s22: 3
    s23: 1
    s21: 2
  s25:
    s22: 3
    s27: 1
#    s21: 2

hostIp:
  s27: 192.168.101.27
  s26: 192.168.108.26

lerIp:
  s23: 192.168.108.23
  s25: 192.168.101.25

path:
  s26:
    s27:
      - s26
      - s23
      - s21
      - s25
      - s27
  s27:
    s26:
      - s27
      - s25
      - s22
      - s23
      - s26
```

# Agent Base Class

- The agent is decoupled from the network operation
  - Greatly simplifies the implementation
  - Does not require to be aware of the actual paths
  - Just need to observe the routing costs to make decisions
- Agent is called passing the flow ID *f* and number of paths |P(f)| (*flow_init*)
- The agent decides the path index (*flow_route*)
- MPLS router implements the selected path

```
class AgentBase():
    #def __init__(self, *args, **kwargs):

    # Agent-specific functions
    def flow_init(self, flowID, nbr_action, costL):
        pass
    def flow_route(self, flowID):
        pass
    def flow_cost(self, flowID, actionL, costL):
        pass
    def flow_close(self, flowID):
        pass

    # Function exposed to the router
    #def getPath(self, rtrName, srcName, dstName):


    # Utility function used by the router
    def getCost(self, fromNode, toNode):
        return 1.0
```

# Route Adaptation

BRISK COMPUTING

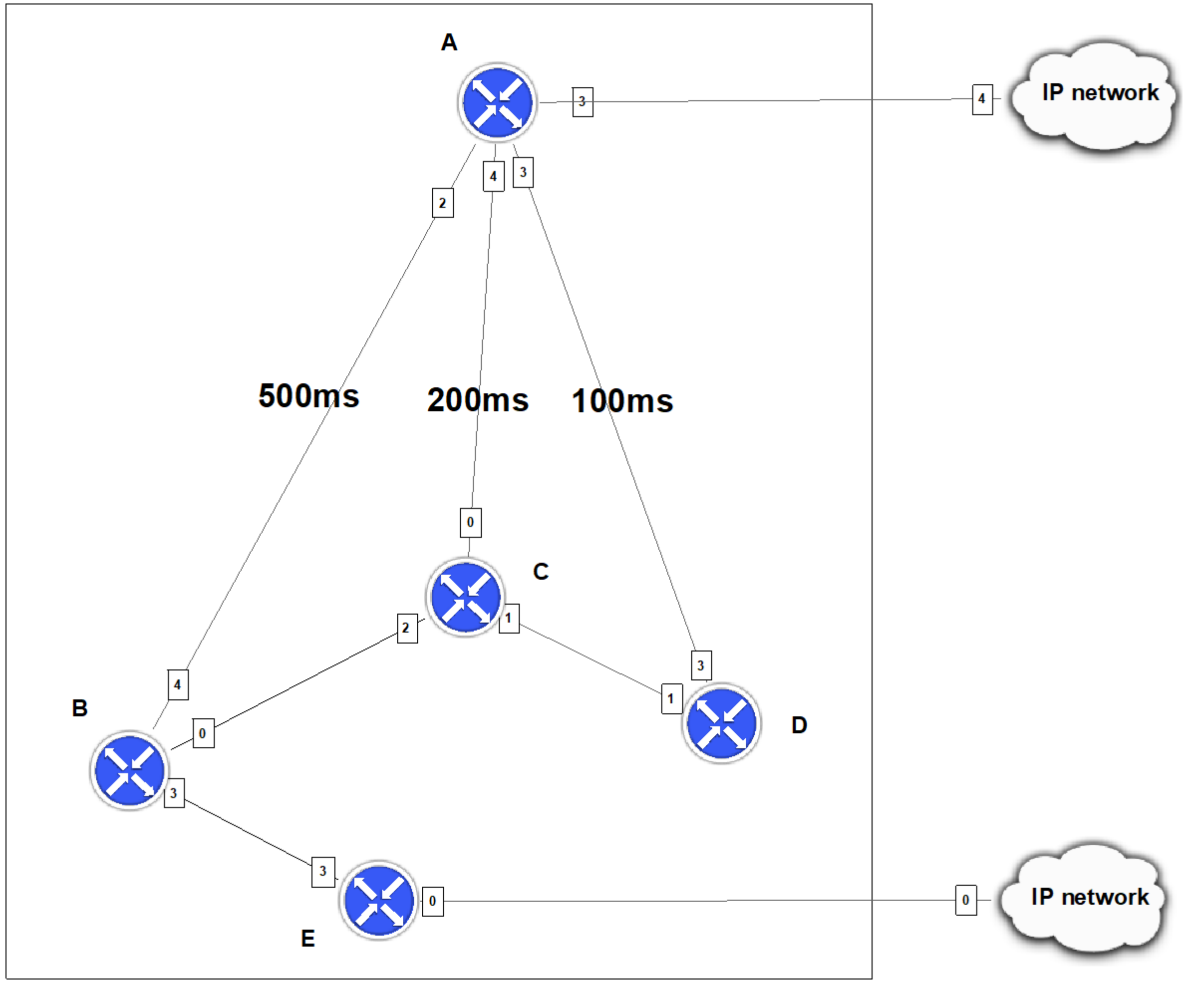
- ❑Link costs are dynamically evaluated by the Monitor. The information becomes part of the knowledge:

```
rlent@s30:~/devel/sdn_exper/Exper1$ curl localhost:8080/switch13/data
{"s22": {"s24": {"delay": 0.001650300464313984, "cnt": 10}, "s25": {"delay": 0.00162362134605784
47, "cnt": 10}, "s23": {"delay": 0.3096590902378288, "cnt": 9}}, "s23": {"s22": {"delay": 0.3378
689080892093, "cnt": 10}, "s21": {"delay": 0.07774153303166254, "cnt": 10}, "s24": {"delay": 0.1
3889510999074178, "cnt": 9}}, "s21": {"s23": {"delay": 0.0015240369358162878, "cnt": 9}, "s24":
{"delay": 0.001529543478949547, "cnt": 9}}, "s24": {"s22": {"delay": 0.0015023722411098483, "cnt
```

- ❑The path cost (negative reward) of the active flows is evaluated after a link cost change
- ❑The agent is informed of the affected flows and new path costs (*flow_cost*)
- ❑The router periodically attempts to modify the path of the active flows (*flow_route*)

# Experimental Setup

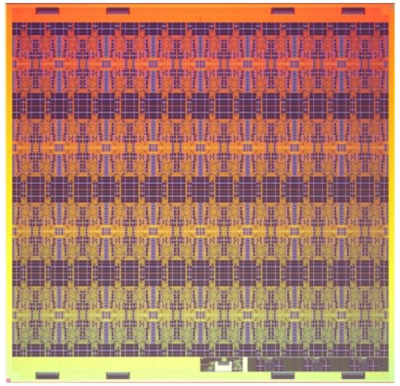
- A high-throughput satellite (HTS) system was emulated in the University of Houston's laboratory facilities.
- One satellite router (node A) connected to four ground stations and routers (nodes B, C, D, E).
- The ground router E is connected to an external IP network. The router implementation was deployed on nodes A, B, C, D, and E with nodes A and E operating as LER and the rest of nodes as LSR.
- One way propagation delay was introduced both for the downlink and the uplink using Linux's Traffic Control (TC).
- The delays for the links A-B, A-C, and A-D were configured with 500ms, 200ms, and 100ms.
- A YAML interface (a network configuration file) was developed to pass the network topology to the controller.



# Reinforcement Learning on Intel Loihi

- ❑ Reinforcement learning method learns
  - ➢ by interacting with its environment and
  - ➢ receiving rewards or punishment from its interactions.
- ❑ Q-learning is one of the basic working principle of Reinforcement learning.
- ❑ Q-learning tries to learn an optimal action-selection policy for any given finite Markov Decision Process.

- ❑ It is used in game playing, control systems, operations research, multi-agent systems, and so on.



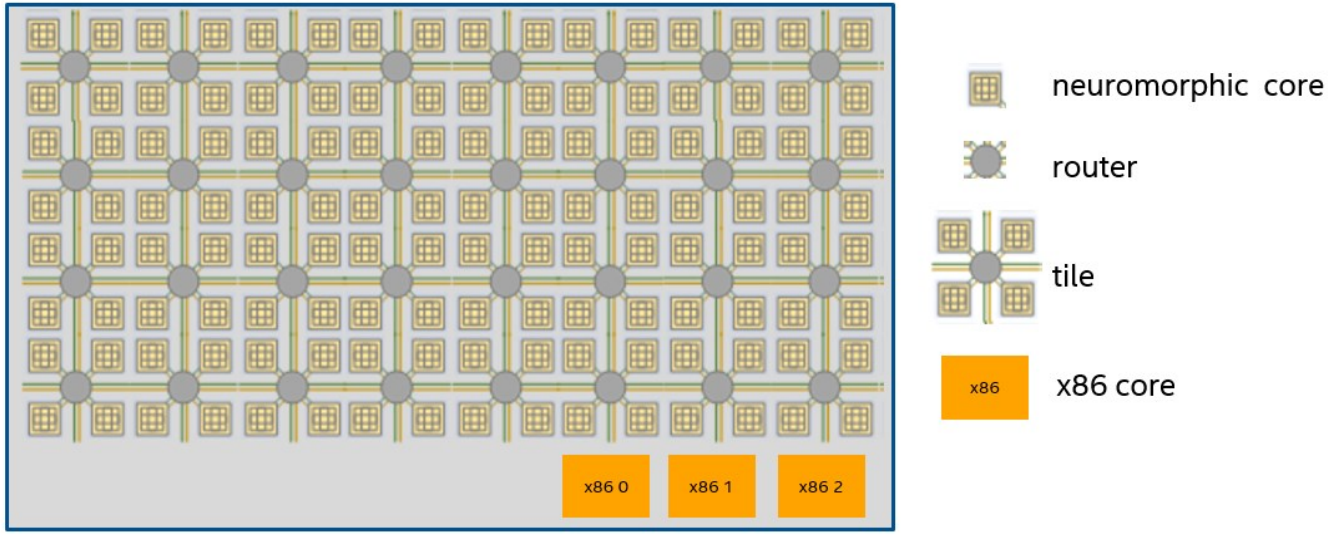
Define Reinforcement Learning Model for Network Routing → Convert the Model into Spiking form using Nengo DL → Keras Models to Nengo Networks → Implement using Nengo Loihi → Implement on Loihi

# Intel's Neuromorphic Processor (Loihi)

BRISK COMPUTING

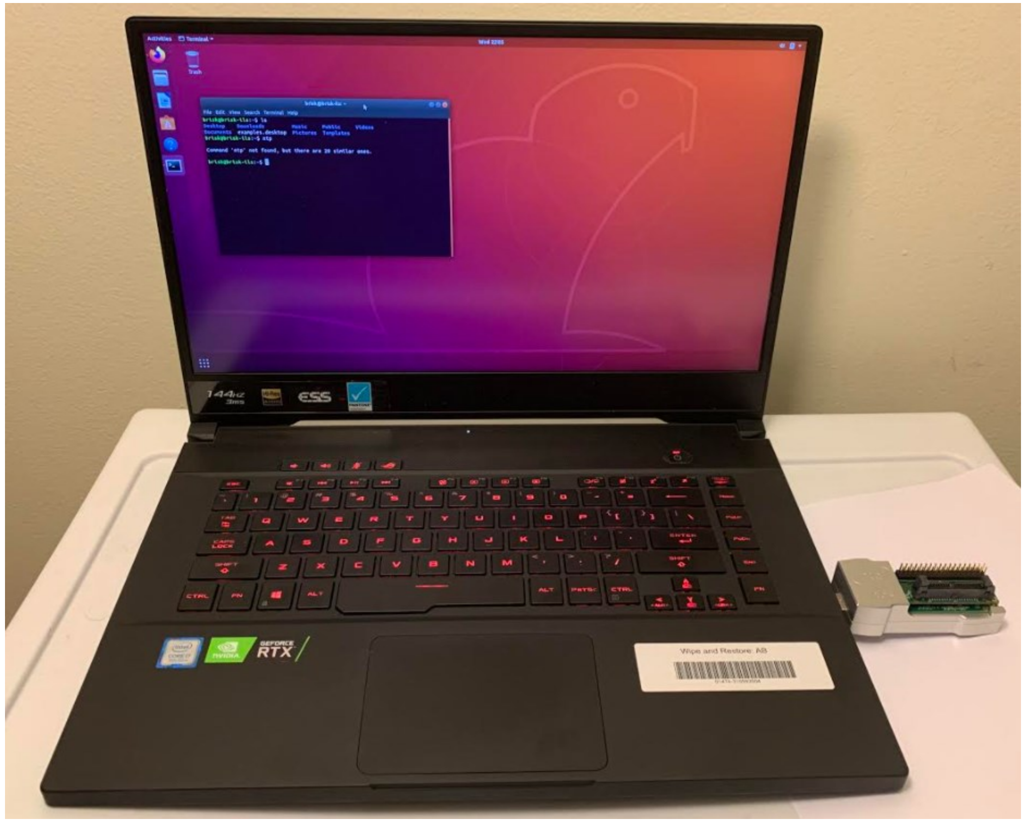
- Each Loihi Chip consists of 3 synchronous x86 cores and 128 neuromorphic cores.

- Loihi is available in several Size:
  - Kapoho Bay (1 or 2 chips)
  - Wolf Mountain (4 chips),
  - Nahuku (32 chips),
  - Pohoiki Beach (64 chips),
  - Pohoiki Springs (764 chips)
- Programming Language:
  - NxSDK,
  - Nengo (Deep learning),
  - Slayer (Deep learning),
  - LAVA

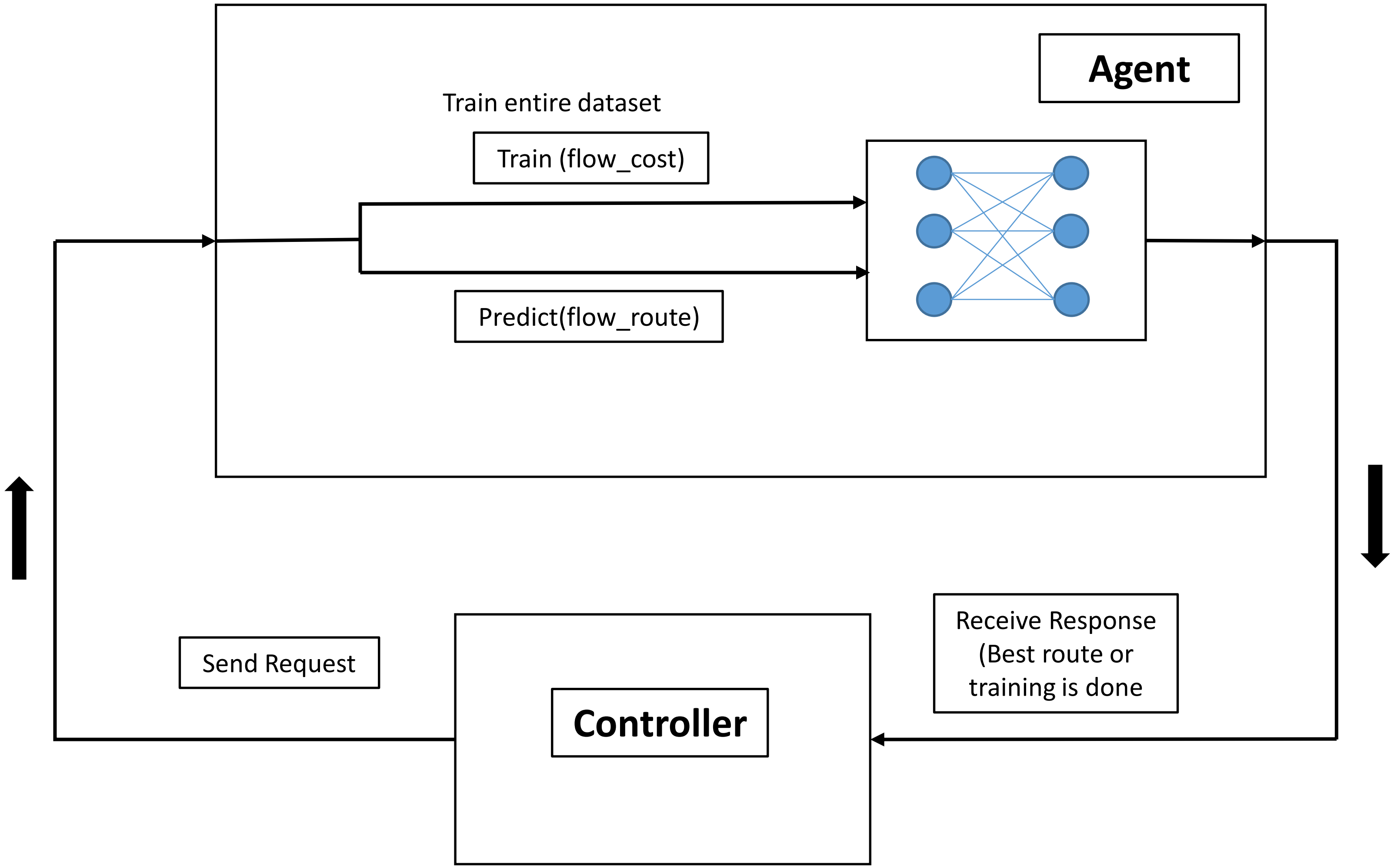


# Loihi Kapoho Bay on Laptop at Brisk

- **Kapoho Bay** is a USB stick form factor that incorporates 1 or 2 Loihi chips.
- With 2 chip Kapoho Bay has 256 neuromorphic cores with 262,144 neurons and 260,000,000 synapses.

# Agent's Structure



# Dataset Processing

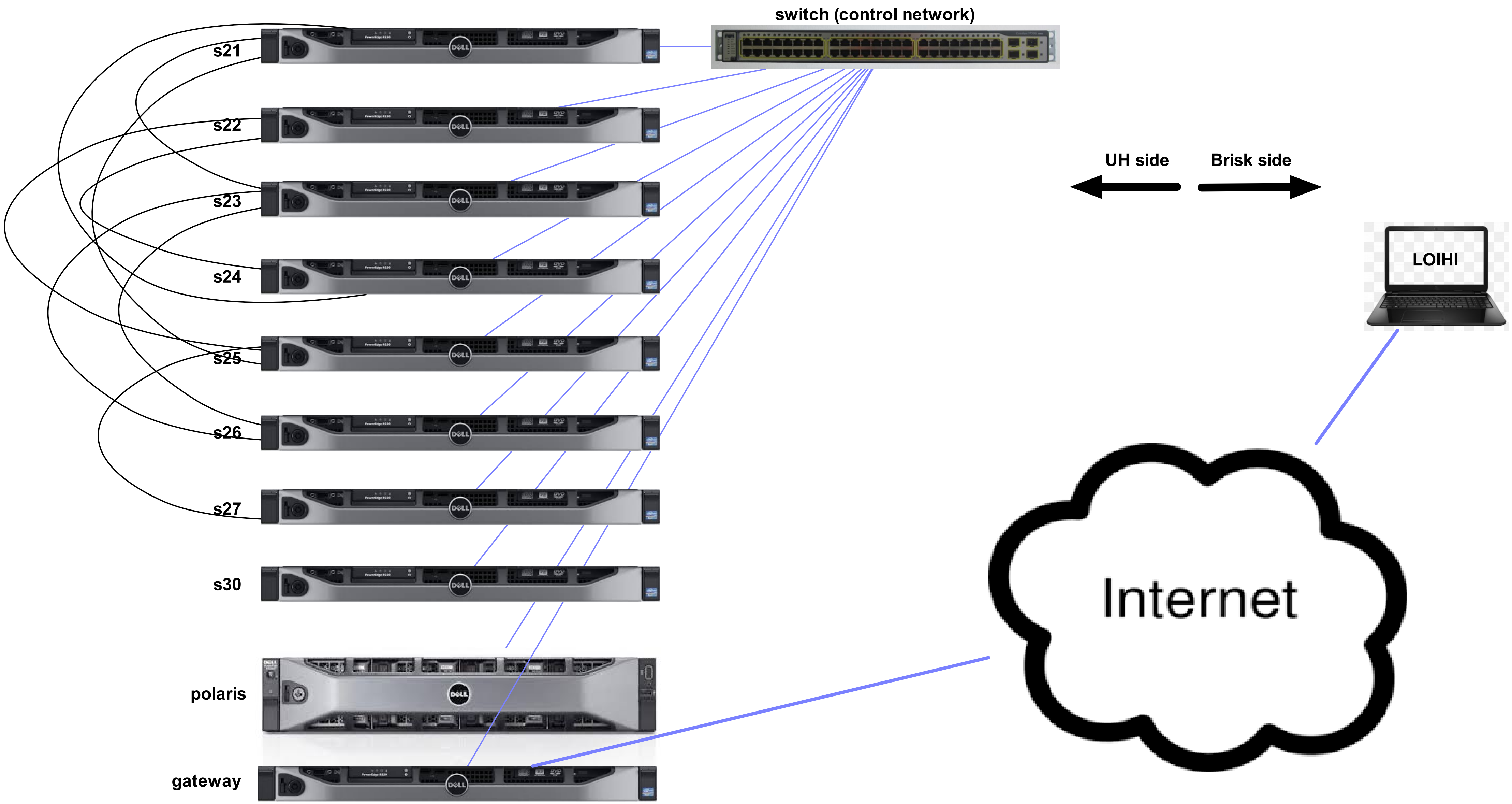
- The input state of each training sample was constructed using the flow ID and the previous timestamps received reward.
- At first, a unique binary 4×4 matrix was created for each flow ID by converting it into a 16bit string using the equation: (round(flow id/$2^{10}$)*10000).

| Flow ID | Binarized representation |
|---|---|
| 1 | 0000000000001010 |
| 2 | 0000000000010100 |

- After that the information of the best previous route was appended with that matrix to convert the input state into an 8×8 matrix.
- In the experiment, only three possible routes are considered.
- Flow IDs are divided into two categories: downlink ('s23') and uplink ('s25') where each category can have maximum 1024 flow IDs

# Distributed Test Environment

BRISK COMPUTING



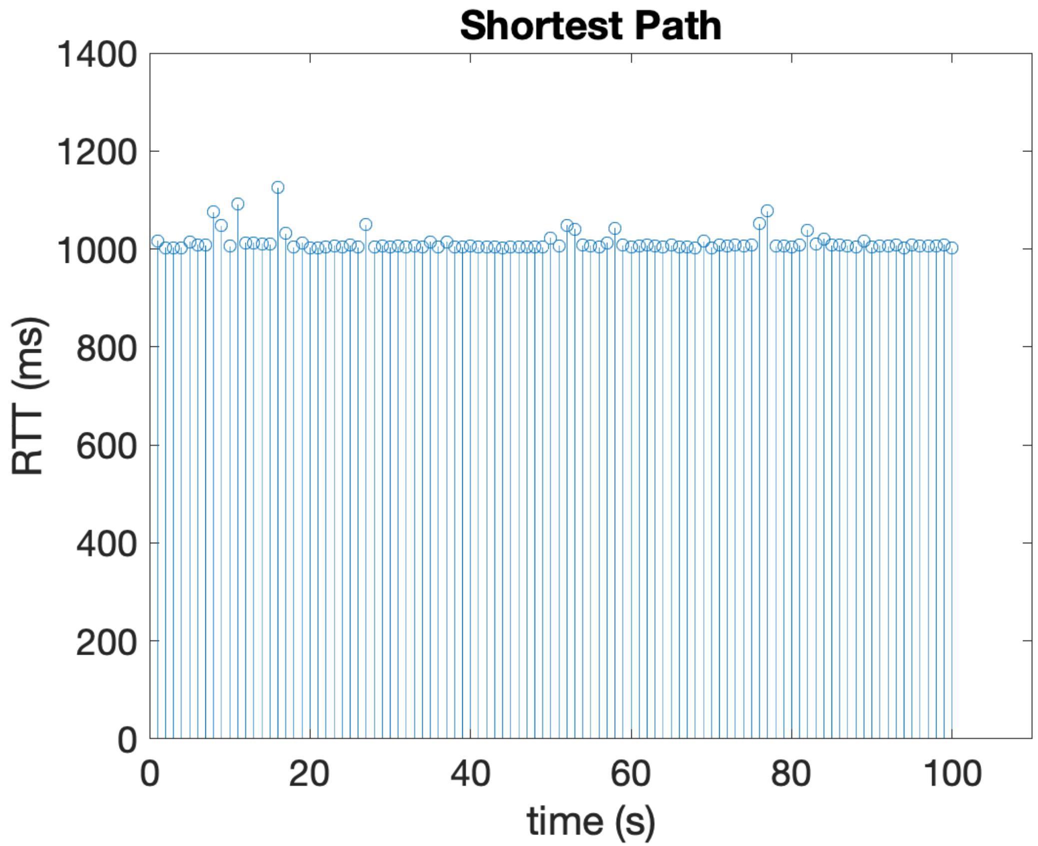
# Results: Shortest Path

BRISK COMPUTING

❑Test: end-to-end ping

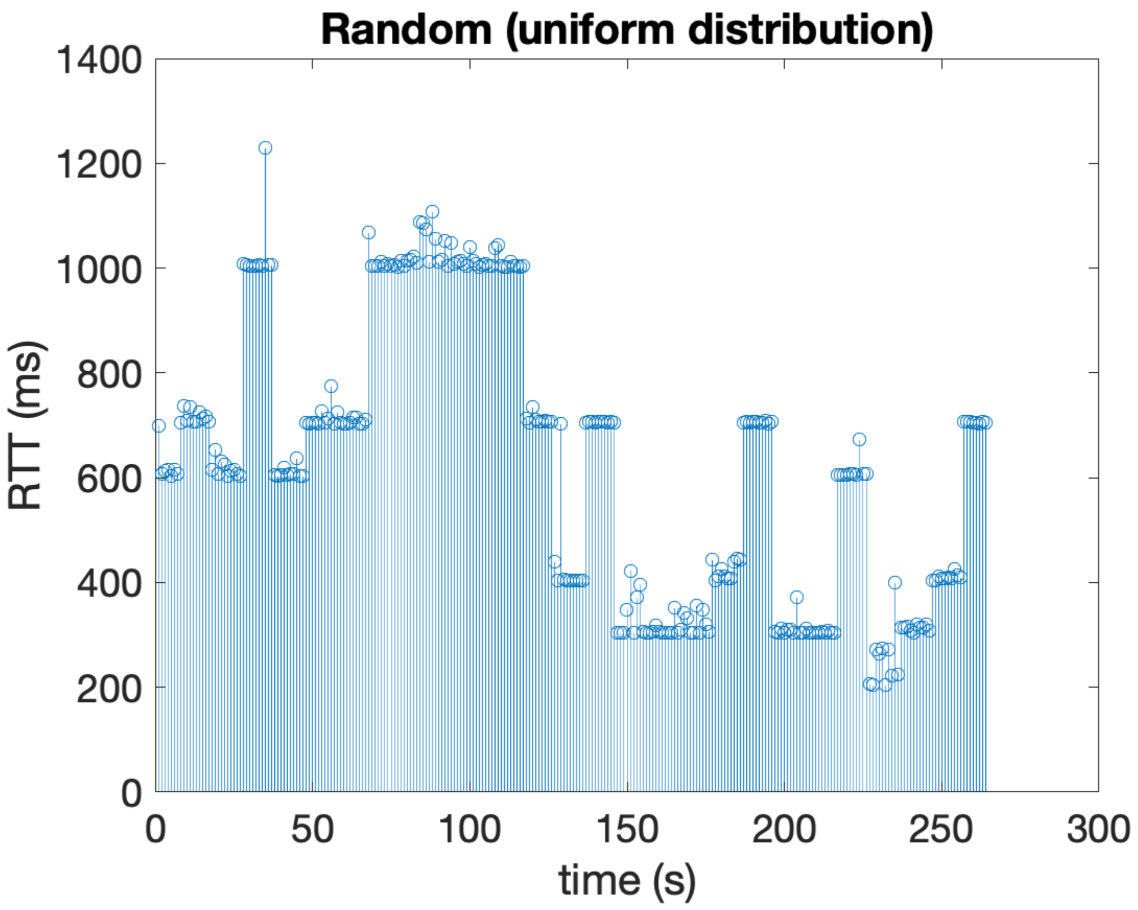
- Link state test rate:  1 pps
- Route update rate: 0.1 route/s (max)

```
class AgentShortestPath(AgentBase):

    def __init__(self, *args, **kwargs):
        super(AgentShortestPath, self).__init__(*args, **kwargs)


    def flow_init(self, flowID, nbr_action, costL):
        pass

    def flow_route(self, flowID):
        return 0                    # paths are sorted by their initial cost

    def flow_cost(self, flowID, actionL, costL):
        pass

    def flow_close(self, flowID):
        pass
```



# Random Path

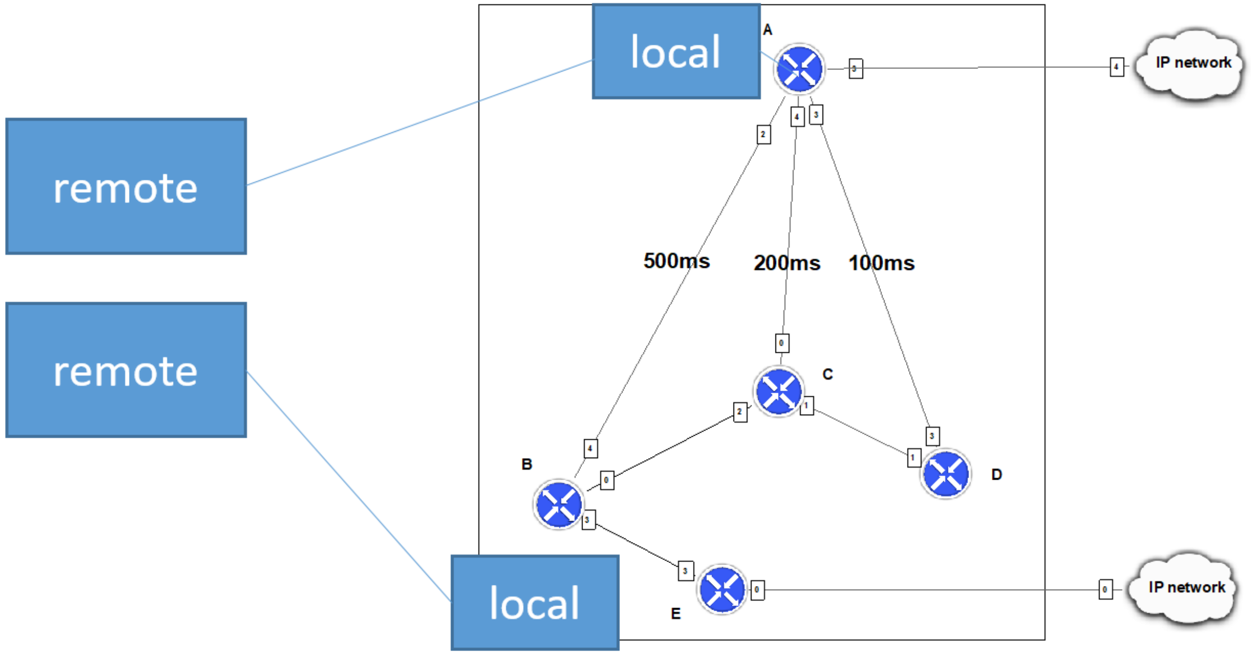
```python
class AgentRandom(AgentBase):

    def __init__(self, *args, **kwargs):
        super(AgentRandom, self).__init__(*args, **kwargs)
        self.nactionD = {}

    def flow_init(self, flowID, nbr_action, costL):
        self.nactionD[flowID] = nbr_action

    def flow_route(self, flowID):
        return random.sample(range(self.nactionD[flowID]), 1)[0]

    def flow_cost(self, flowID, actionL, costL):
        pass

    def flow_close(self, flowID):
        pass
```

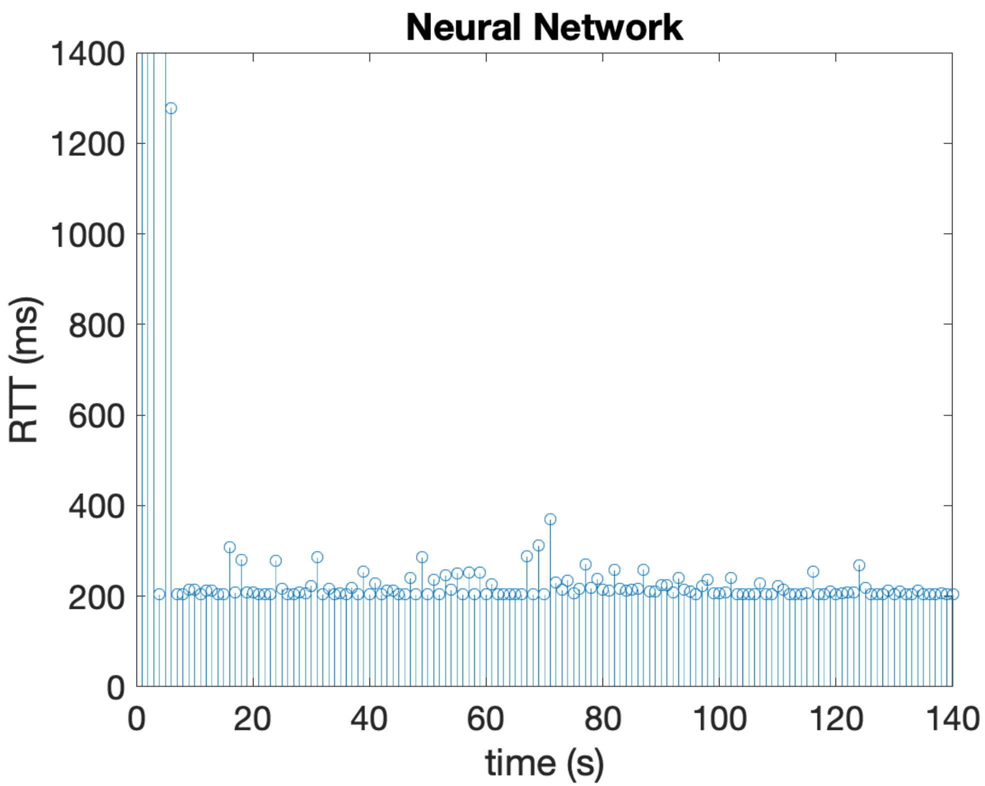


# Neural Networks

Agent:

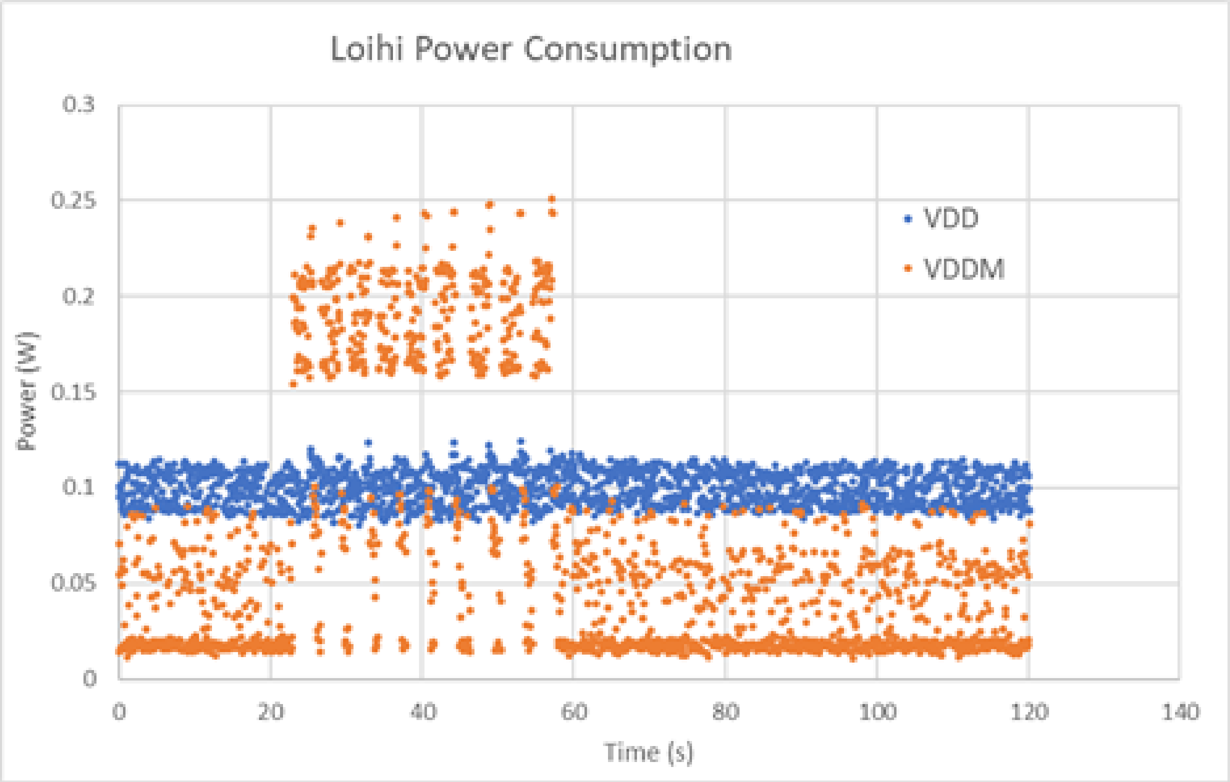
- NN agent consists of a local and a remote component
- Local component: interfaces with the Ryu controller and the proposed SDN/VNF architecture
- Remote component: implements NNs with Keras and Tensorflow
- RESTful communication between the 2 components



# Energy Calculation

- ❑Application ran for 10 continuous operation (each operation took 12 seconds).
- ❑Power was measured for 30 seconds while running the routing application.
- ❑The energy consumption reaches around 7.5 Joule (0.25W × 30s).



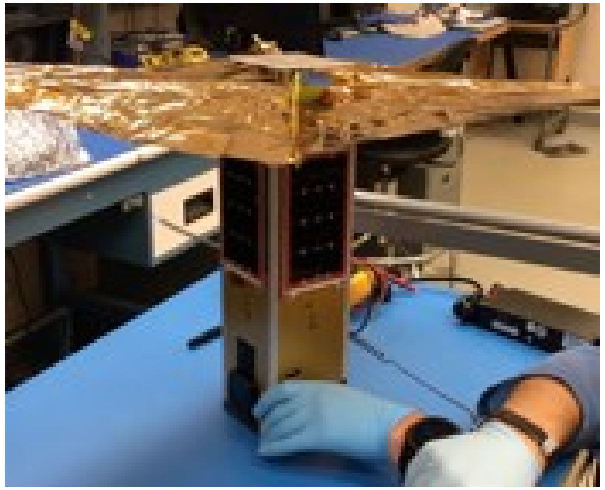

- VDD: Neuro cores, embedded Lakemont CPU, mesh router, FPIO/PIO (programmed input-output) protocol logic, etc. (everything else that consumes power on the chip) are expressed as VDD and
- VDDM: VDDM shows the power consumption by the SRAM memories

# Space Exploration for Future Technology

BRISK COMPUTING

- On January 13th, 2022, the TES-13 CubeSat launched on a Virgin Orbit launch vehicle flight and has recently successfully finished the successful Phase 1 Operations.
- The TES-13 is based on a 3U nano-sat design.
- NASA launched a Loihi Kapoho Bay USB unit hosted by an Up Board in a CubeSat (TES-13) to test the machine learning algorithms on the Loihi in a space environment. Loihi based SDN application was selected for this launch.

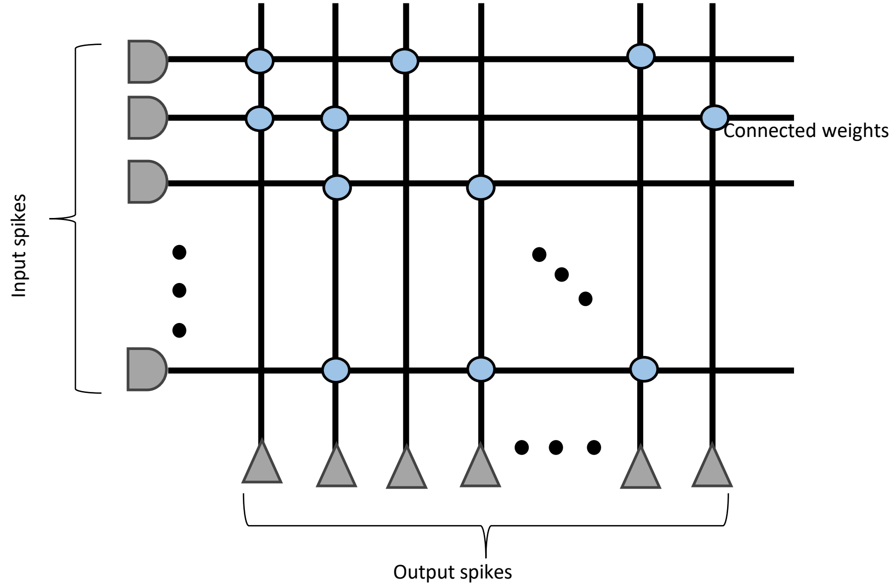
The TES-13 was launched on the Virgin Orbit Launcher 1



The TES-13 in preparation for integration into the dispenser

# Modified SDN Implementation

- ❑ Nengo-DL (deep learning library of Nengo) could not run on the Up Board due to the limited supported libraries (e.g., vector assembly instruction is not supported) by the Pentium processor on Up Board.
- ❑ As a result, a new classification-based spiking feed-forward network (a two-layer perceptron network) was developed for the Loihi using Intel's NxSDK that performed the core functionality of the routing.
- ❑ In the network, when the dot product of the input signals and the weights of a given neuron exceeds the predefined threshold, then the neuron will send a spike to the following layer.
- ❑ The networks weights were optimized in MATLAB and stored in the Loihi neuromorphic cores.



# Modified SDN Implementation

BRISK COMPUTING

- This simplified version had no learning (it predicted the best route all the times).
- Due to memory limitation, instead of writing all the neurons firing patterns, sum of all the spikes generated by the input neurons were stored for each execution. The final output of this approach is an 87-character string of numbers that would be transmitted to earth from the CubeSat to indicate successful execution.
- This output with spaces added in to clarify what the different fields are is shown below:

Output spike

8 2 1011101111 100 2 1101111110 001 2 1101101110 010 8 2 1011101111 100 2 1101111110

001 2 1101101110 010 8

- First '8' shows the program ran until the first Loihi call.
- 2 - number of input neurons that spiked
- 1011101111 - spiking pattern of 10 middle layer neurons
- 100 - spiking pattern of 3 output neurons
- 2 - start of a repeating pattern for the next execution
- The second '8' shows that the program completed the multicore Loihi calls (the first 3 runs).
- The third '8' shows all lines of code were executed without error.

# Conclusion

BRISK COMPUTING

❑ Implemented SDN application on neuromorphic hardware in space.

**Future work:**

❑ Execute developed learning enabled SDN application into space.

# Backup

# Loihi Connection

Connected to server (laptop)

```
(python3_venv_clone) (loihi_env_clone) brisk@brisk-ils:~/Documents/AnacondaExamples/server_network_nengo$ lsusb -t
/:  Bus 04.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/4p, 10000M
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 480M
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/8p, 10000M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/16p, 480M
    |__ Port 4: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 3: Dev 8, If 1, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 3: Dev 8, If 0, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 1: Dev 4, If 0, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 1: Dev 4, If 1, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 4: Dev 9, If 0, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 2: Dev 6, If 0, Class=Vendor Specific Class, Driver=, 480M
        |__ Port 2: Dev 6, If 1, Class=Vendor Specific Class, Driver=, 480M
    |__ Port 5: Dev 3, If 1, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 5: Dev 3, If 2, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 5: Dev 3, If 0, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 10: Dev 5, If 2, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 10: Dev 5, If 0, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 10: Dev 5, If 1, Class=Human Interface Device, Driver=usbhid, 12M
    |__ Port 14: Dev 7, If 0, Class=Wireless, Driver=btusb, 12M
    |__ Port 14: Dev 7, If 1, Class=Wireless, Driver=btusb, 12M
(python3_venv_clone) (loihi_env_clone) brisk@brisk-ils:~/Documents/AnacondaExamples/server_network_nengo$ `python3 -c "import nxsdk; print(nxsdk.__path__[0])"`/bin/x86/kb/lakemont_driver --test-fpio
Using Kapoho Bay serial number 434
test_fpio_loopback: chain=0 chips=2 blocks=1000 time=10436us => 1.53315Mb/s
(python3_venv_clone) (loihi_env_clone) brisk@brisk-ils:~/Documents/AnacondaExamples/server_network_nengo$
```

# Output from Loihi

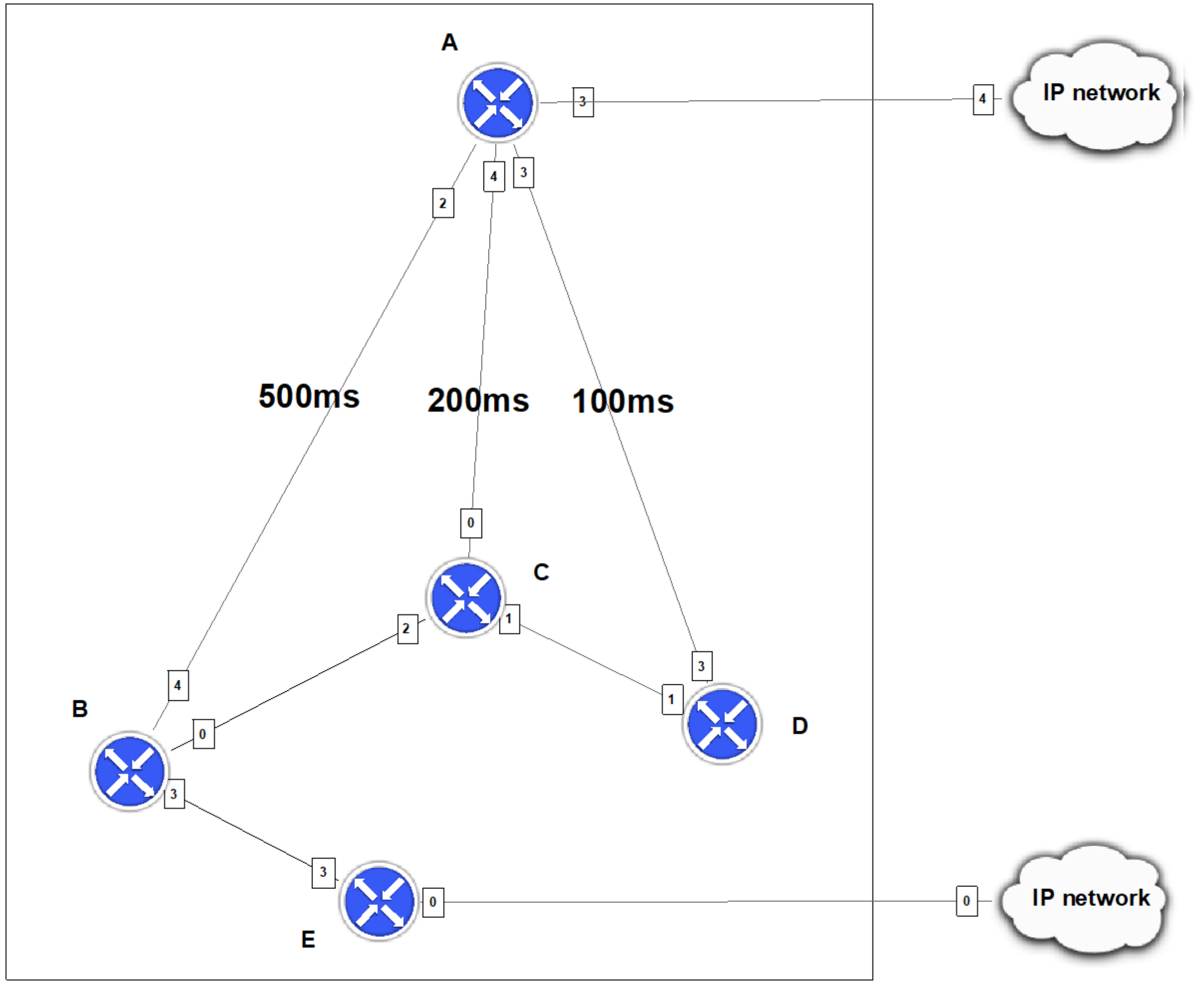
BRISK COMPUTING

```
1612379556.617951 root         DEBUG    Entered run_network_onLoihi method::::: [keras_server_mem.py:179 run_network_onLoihi]
Build finished in 0:00:00
|#                                                                        Optimizing graph: creating signals                                                          | 0:00:001
612379556.642543 root         DEBUG    partitions [tensor_graph.py:1066 create_signals]
1612379556.642638 root         DEBUG
 || |||||||||||||||||| |  [tensor_graph.py:1068 create_signals]
Optimization finished in 0:00:00
Construction finished in 0:00:00
INFO:DRV:  Connecting to 127.0.0.1:42221
1612379557.024463 DRV          INFO     Connecting to 127.0.0.1:42221 [host_coordinator.py:75 connect]
INFO:DRV:      Host server up.............Done 0.20s
1612379557.026053 DRV          INFO         Host server up.............Done 0.20s [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Encoding axons/synapses.....Done 1.34ms
1612379557.028196 DRV          INFO         Encoding axons/synapses.....Done 1.34ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Compiling Embedded snips....Done 0.12s
1612379557.150447 DRV          INFO         Compiling Embedded snips....Done 0.12s [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Booting up..................Done 2.95s
1612379560.099556 DRV          INFO         Booting up..................Done 2.95s [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Encoding probes............Done 0.15ms
1612379560.100240 DRV          INFO         Encoding probes............Done 0.15ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Transferring probes.........Done 1.24ms
1612379560.119232 DRV          INFO         Transferring probes.........Done 1.24ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Configuring registers.......Done 7.21ms
1612379560.126871 DRV          INFO         Configuring registers.......Done 7.21ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Transferring spikes.........Done 2.68ms
1612379560.129928 DRV          INFO         Transferring spikes.........Done 2.68ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Executing...................Done 0.09s
1612379560.224451 DRV          INFO         Executing...................Done 0.09s [nxlogging.py:266 timedContextLogging]
INFO:DRV:      Processing timeseries.......Done 0.79ms
1612379560.225633 DRV          INFO         Processing timeseries.......Done 0.79ms [nxlogging.py:266 timedContextLogging]
INFO:DRV:  Executor: 30 timesteps.........Done 0.11s
1612379560.225747 DRV          INFO     Executor: 30 timesteps.........Done 0.11s [nxlogging.py:266 timedContextLogging]
Predict Outcome_L is: [[0.49776891 0.54278774 0.62189235]]
Predict Label_L is: [2]
INFO:HST:  Using Kapoho Bay serial number 434
1612379560.236293 HST          INFO     Using Kapoho Bay serial number 434 [subprocess_logging_handler.py:36 run]
INFO:HST:  Args chip=0 cpu=0 /home/brisk/python3_venv_clone/lib/python3.5/site-packages/nxsdk/driver/compilers/../../../temp/1612379557.0283532/launcher_chip0_lmt0.bin --chips=1 --remote-relay=0
1612379560.236421 HST          INFO     Args chip=0 cpu=0 /home/brisk/python3_venv_clone/lib/python3.5/site-packages/nxsdk/driver/compilers/../../../temp/1612379557.0283532/launcher_chip0_lmt0.bin --chips
=1 --remote-relay=0  [subprocess_logging_handler.py:36 run]
INFO:HST:  Lakemont_driver...
1612379560.236482 HST          INFO     Lakemont_driver... [subprocess_logging_handler.py:36 run]
INFO:HST:  chip=0 cpu=0 halted, status=0x0
1612379560.236556 HST          INFO     chip=0 cpu=0 halted, status=0x0 [subprocess_logging_handler.py:36 run]
1612379560.245826 root         DEBUG    Model output: 1 [keras_server_mem.py:410 flow_route]
1612379560.246007 root         DEBUG    Loihi output: [2] [keras_server_mem.py:411 flow_route]
```

# Task 3: Integration and Validation

BRISK COMPUTING

- Goal: To experimentally demonstrate a LOIHI-based SDN/VNF routing approach
- Implementation:
  - Ryu controller
  - Open VSwitch (OVS)
  - MPLS routing provided through a VNF
  - OVS nodes are virtualized with a 1:1 mapping of ports to physical interfaces
  - Hardware OpenFlow switches to be tested



# Demo

BRISK COMPUTING