Verifying the CPA Networking Stack using SPIN/Promela

Kevin Chalmers and Jon Kerridge

Edinburgh Napier University

Breakdown

- Introduction and Motivation
- CPA Networking Stack
    - Architecture
    - Operation
- SPIN Model of the CPA Networking
    - Processes
    - Architecture
- Results
- Conclusions and Future Work

Motivation

- CPA Networking Stack developed from JCSP Networking
  - net2 package
- Original JCSP Networking had poor error handling
  - Errors in the stack not sent to application layer
- Verify CPA Networking Stack operates under certain conditions
  - Bufferring
  - Network failure

SPIN/Promela

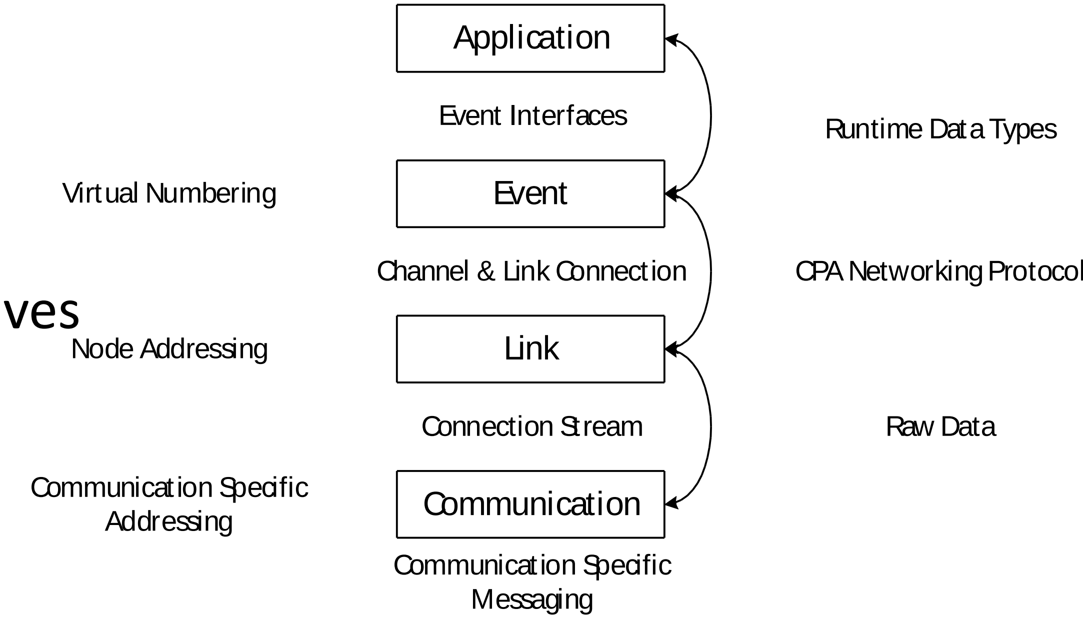
- SPIN (Simple Promela INterpreter) provides state space checking
  - Assertion checking
  - Deadlock
  - Liveness
- Language to build SPIN models is Promela (PROcess MEta LAnguage)
- Similar semantics to CSP
  - Components as processes
  - Processes communicate via channels
  - Choice between events
- Provides channel mobility (CPA Networking Stack currently relies on channel mobility internally)

CPA Networking Stack

- CPA Networking Stack developed from JCSP Networking
- Goal is to provide a method to allow multiple platform / framework communication in a transparent CPA manner
  - Networked channels
  - Networked barriers
- Development of standard components and protocol
- Take two views
  - Layered architecture
  - High-level component architecture

Layered Architecture

- Application layer
  - User level processes
- Event layer
  - Networked synchronization primitives
- Link layer
  - Connections to other nodes
- Communication layer
  - Underlying I/O mechanism



Application

Event Interfaces

Runtime Data Types

Virtual Numbering

Event

Channel & Link Connection

CPA Networking Protocol

Node Addressing

Link

Connection Stream

Raw Data

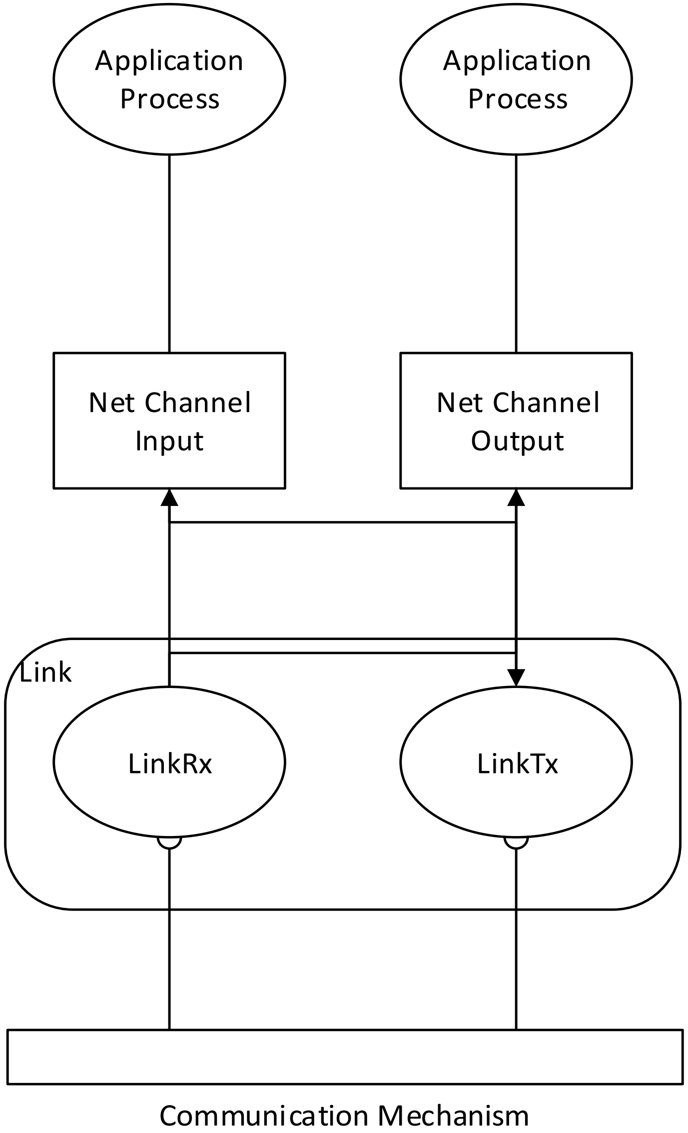
Communication Specific Addressing

Communication

Communication Specific Messaging

High Level Architecture

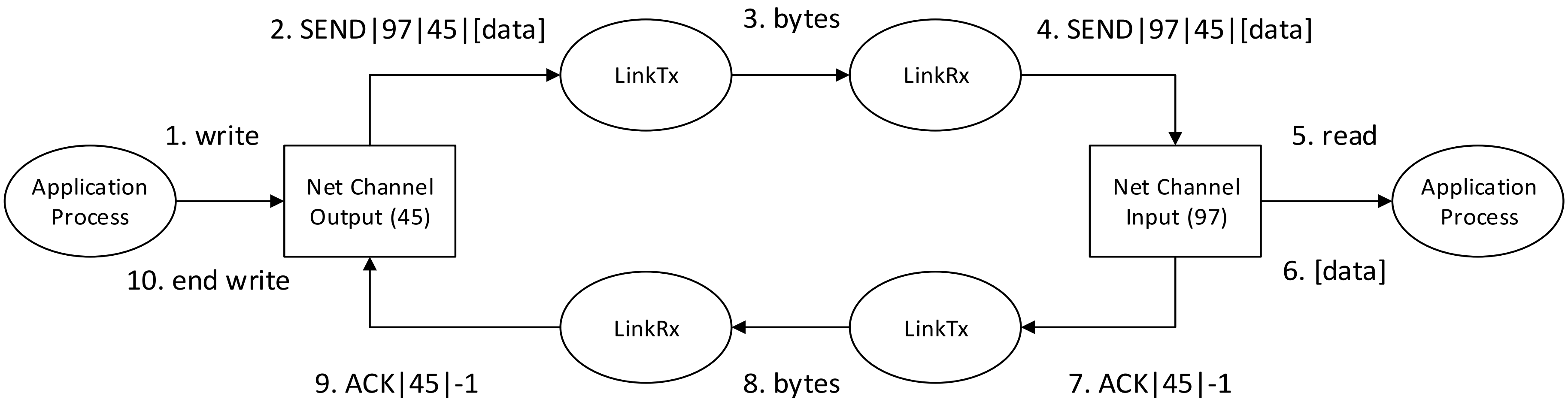
- Link
    - LinkTx for outgoing messages
    - LinkRx for incoming messages – protocol implemented here
- Networked channels
    - Output provides a writing end
    - Input provides a reading end
- Other components for management, barriers



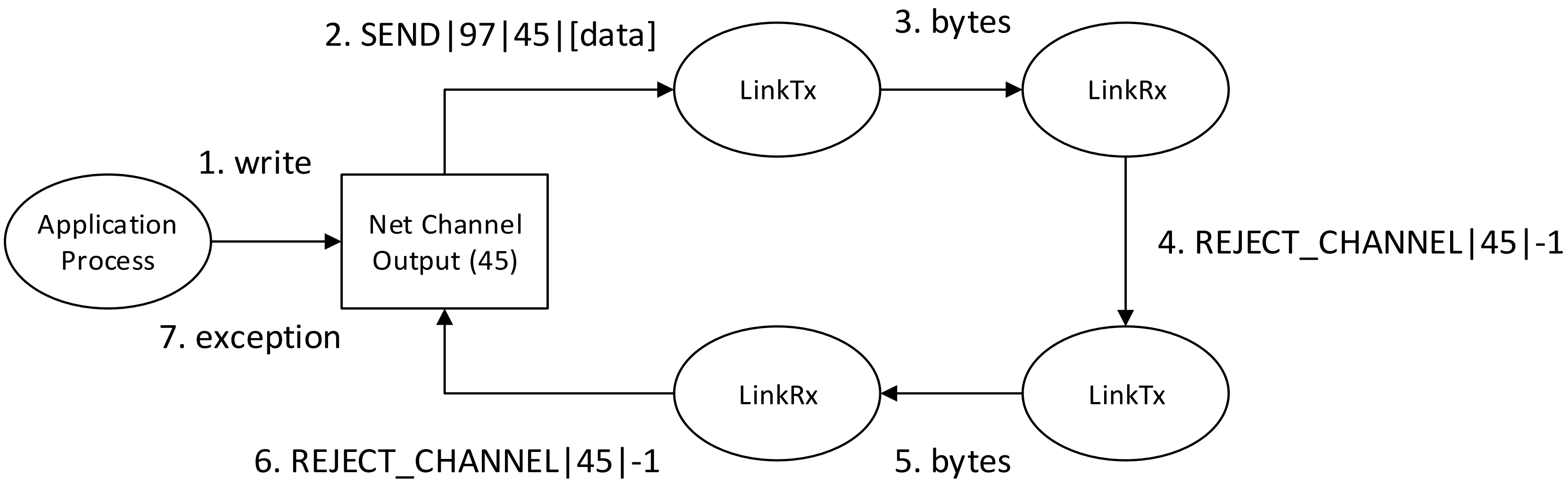
Protocol

- Message defined by a triple (with possible data load)
  - (<type>, <attr1>, <attr2>, [<data>])
- Basic channel messages
  - (SEND, <dest>, <source>, <data>)
  - (ACK, <dest>, null)
  - (REJECT_CHANNEL, <dest>, null)
  - (POISON, <dest>, <strength>)
  - (LINK_LOST, <null>, <null>)
  - (ASYNC_SEND, <dest>, <source>, <data>)

SEND/ACK Operation



SEND/REJECT Operation



SEND/LINK_LOST

- One of the biggest issues in JCSP Networking
- Link failure caused resources to remain and messages to disappear
- LINK_LOST message now informs all outgoing channels of link failure
- Two possibilities
  - Prior to a write, link goes down. SEND message immediately replied with LINK_LOST
  - Mid-write link goes down. All output channels connected to link are sent LINK_LOST

Building a SPIN Model of CPA Networking

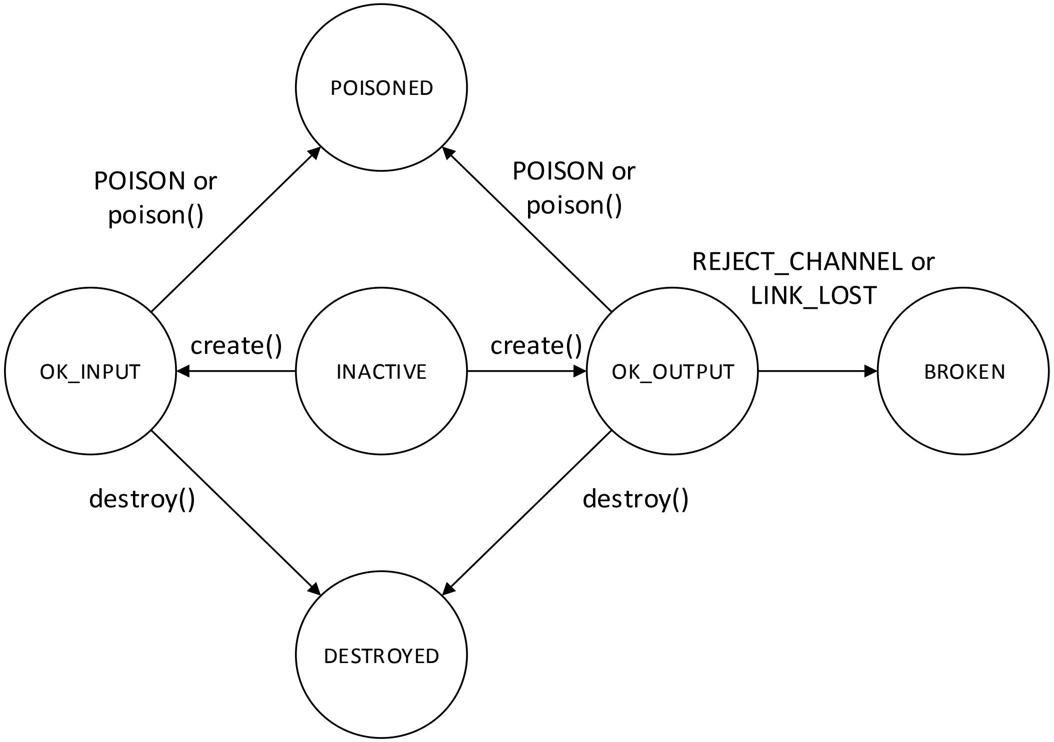
- Only five messages of interest from protocol
  - ASYNC_SEND cannot be checked as sender waits for no ACK – infinite state space
- Promela uses mtype to define message types

```
mtype = { SEND , ACK , REJECT_CHANNEL , POISON , LINK_LOST };
```

Channel States

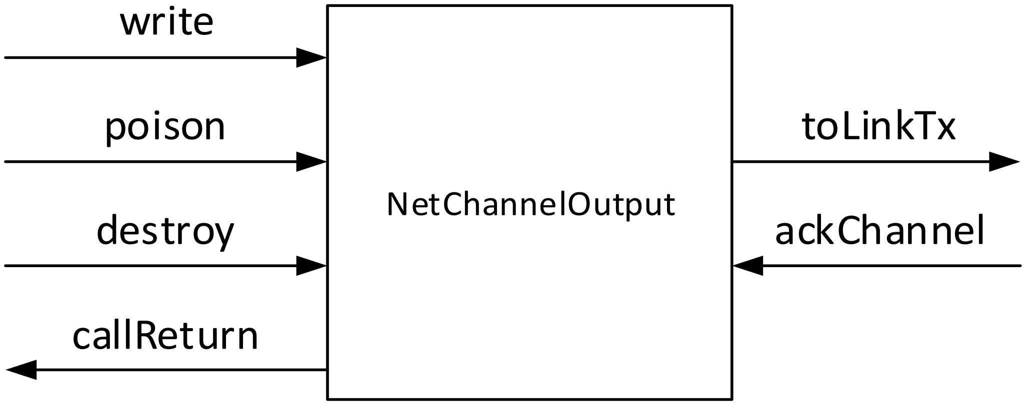
- INACTIVE
- OK_INPUT
- OK_OUTPUT
- POISONED
- DESTROYED
- BROKEN

```
typedef CHANNEL_DATA
{
		byte vcn ;
		byte state = INACTIVE ;
		chan toChannel ;
};
```



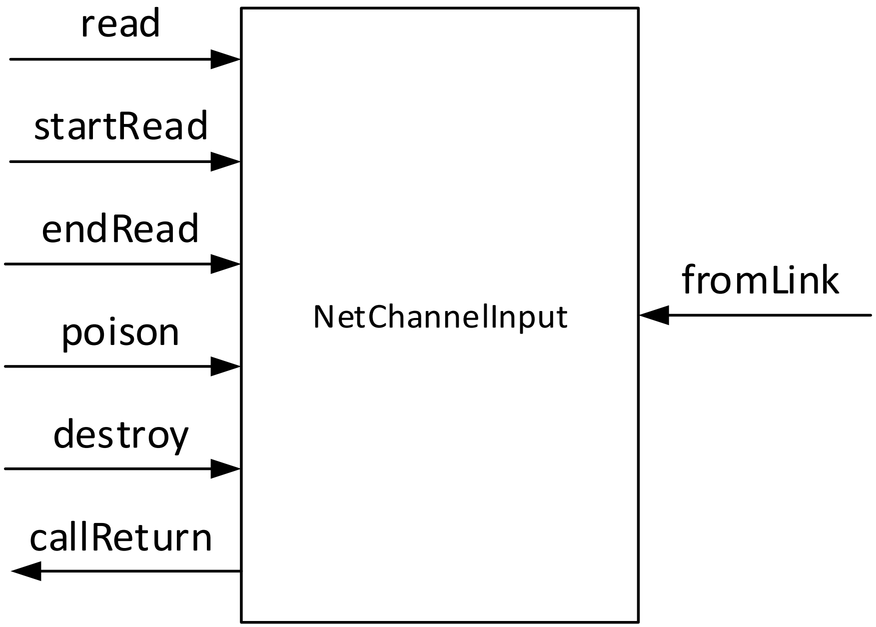
NetChannelOutput

- Use channels to simulate method calls
- Three operations
  - Write
  - Poison
  - Destroy
- NetChannelOutput connected to a LinkTx
- Incoming acknowledgement channel



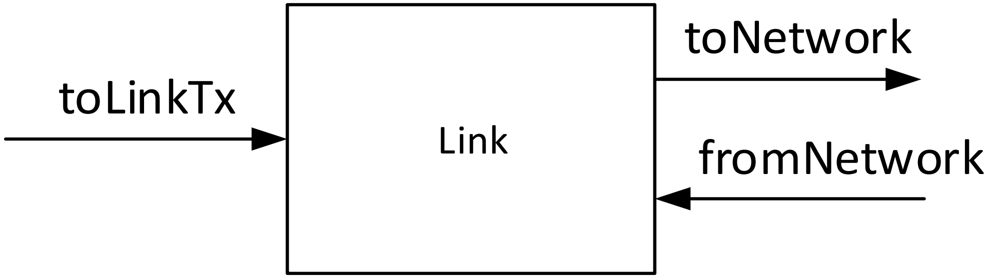
NetChannelInput

- Five operations
  - Read
  - Start Read and End Read
    - Extended rendezvous
  - Poison
  - Destroy
- NetChannelInput has an incoming channel for messages

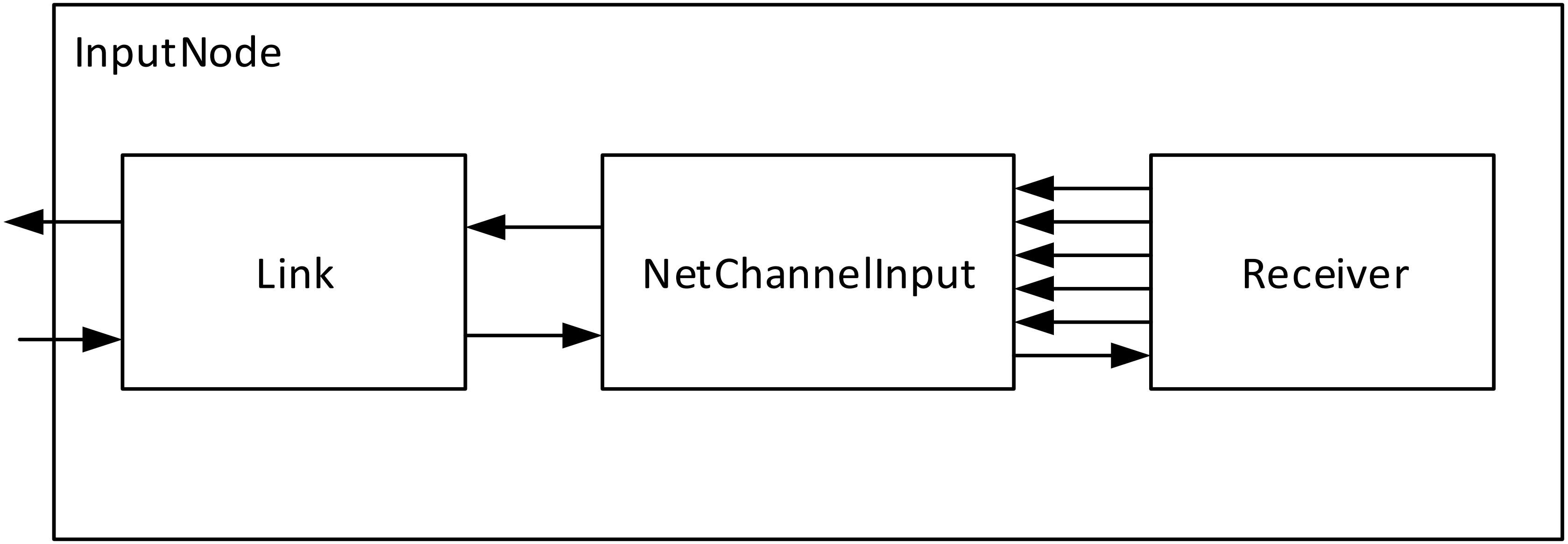


Link Process

- Link contains two sub-processes
  - LinkTx
  - LinkRx – see paper for full Promela code
- Incoming link from event processes
- Connection to the network



InputNode



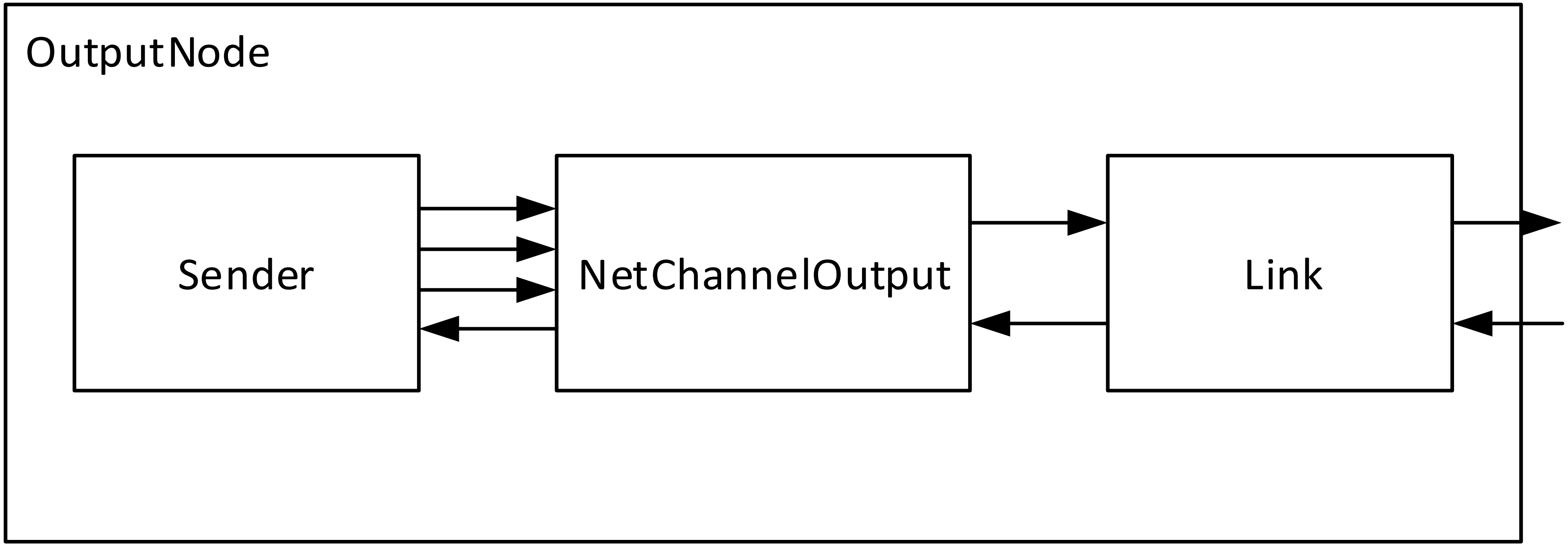
InputNode

Link

NetChannelInput

Receiver

OutputNode

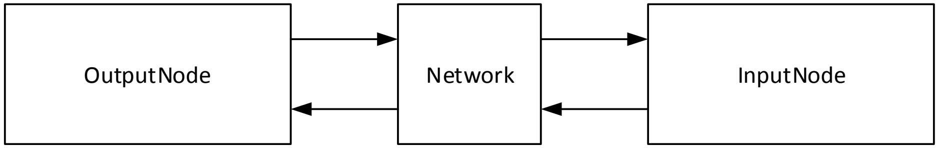


Network Process

- Network process simply forward messages from the InputNode to the OutputNode and vice-versa
- To simulate failure, the Network process can non-deterministically fail
    - See paper for Network process code
- Sending and receiving modelled as atomic – the underlying communication mechanism is assumed to deal with incomplete messages
    - Exceptional behaviour

SPIN Model of CPA Networking

- Model has one OutputNode connected to one InputNode
- The OutputNode can have multiple output channels
- InputNode channel has a buffer
  - Discussed later
- Flag used to determine link failure



Initial Findings

- Single NetChannelOutput connected to a single NetChannelInput with single space buffer successful
  - Basic assumption
  - Link informing NetChannelOutputs of link failure solves link failure problems
- Original JCSP Networking did not lock state of a networked channel
  - Never experienced but would lead to a failed channel being sent a message and no error raised
- State of a channel is now locked – no race hazard!

Verifying the Model - Assumptions

- CPA Networking works on the premise that for every connected network output to a network input, one space is required in the input channel buffer
  - For implementation purposes, a channel has an “infinite” buffer
- To check this, we need to examine the relationship between the number of connected outputs to a network input and the buffer size

Results

NUMBER_OUTPUTS	1	2	3	4
BUFFER_SIZE				
0	FAIL	FAIL	FAIL	FAIL
1	3.06×10^5 *states* 351 *depth*	FAIL	FAIL	FAIL
2	2.78×10^5 *states* 351 *depth*	3.71×10^7 *states* 3264 *depth*	FAIL	FAIL
3	2.78×10^5 *states* 351 *depth*	3.71×10^7 *states* 3264 *depth*	PASS*	FAIL
4	2.78×10^5 *states* 351 *depth*	3.71×10^7 *states* 3264 *depth*	PASS*	PASS*

Conclusions

- CPA Networking Stack is deadlock free even under network failure
- Removed the lack of state protection in the original JCSP implementation
- Buffer size has a relation to number of incoming networked outputs
    - Infinite buffer should ensure deadlock freedom

Future Work

- Really need to show that the networked channel behaves as a standard channel
  - Refinement check
- SPIN doesn’t support refinement checks
  - Temporal logic capabilities
  - Simplify the model and check – but would remove most behaviour
- Current plan is to move to a networking stack that can sit atop MPI
  - Reengineering and further verification would be required

Questions?