24th Sep 2021

LABs on RELIABLE BROADCAST

Lorenzo Ghiro

lorenzo.ghiro@unitn.it

Outline / Goals

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

1. Simple Simulation of Best-Effort Broadcast (BEB)
    1. Simple Node and Channel Failures simulation
    2. See effects of failures on BEB
2. Discrete Event Simulation of R-Broadcast
    1. Architecture of a minimal-DES
    2. Study/inspect/debug a distributed protocol under DES

Simulate BEB

How we simulate BEB:

1. Create a complete graph
2. Define Node and Channel Failure

   PARAMETRIZED models
3. **BEB from pseudo to python!**

Once we have the above implemented...

1. Verify BEB properties under "No Failure" and "With Failure" setting
2. Visualize BEB under the two settings :)

What kind of underlying network?

- Complete graph
  - Every process can communicate with every other process
  - A routing substrate realizes this abstraction

Best-effort broadcast protocol executed by p

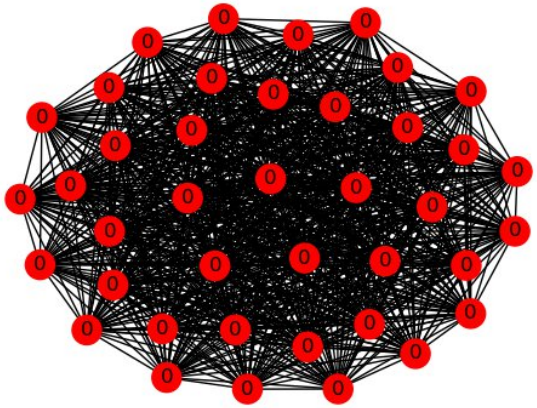
```
upon B-broadcast(m) do
    foreach q ∈ Π do
        send m to q

upon receive(m) do
    B-deliver(m)
```

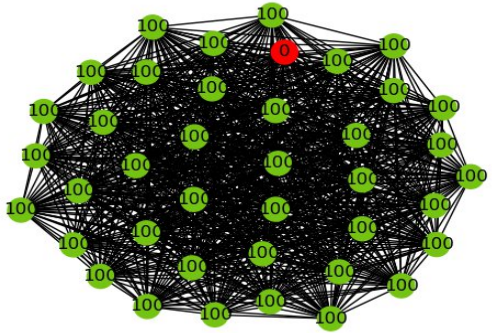
Wishful Output

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

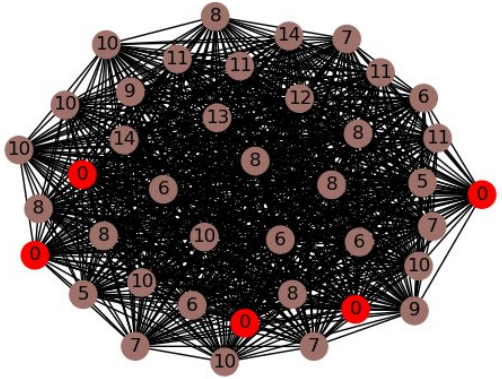
BEFORE BROADCAST



WITHOUT FAILURES



WITH FAILURES



Useful tools

https://networkx.org

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks

Now download the code template and... write what is missing!

Please, do not download solutions now, make a try on your own first :)

BEB solution and test

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

```python
def beb(G):
    # Choose initial node/process (called p)
    p = random.choice(list(G.nodes()))

    # Random color as message, we draw in that color nodes that deliver the message
    rc = '#%02x%02x%02x' % tuple(np.random.randint(256, size=3))

    for n in set(set(G.nodes()) - {p}):
        print("Sending message from {} --> {}".format(p, n))
        # Simulate crash during sending operation
        if random.random() > G.nodes[p]["crashP"]:
            send(G, p, n, rc)
        else:
            print("Node {} crashed!!!".format(p))
            return
```

```python
def send(G, source, dest, color):
    # Simulate channel conditions!

    # First simulate loss
    if random.random() < G[source][dest]['lossP']:
        print("\tLost packet over {}--{} link".format(source, dest))
        return

    # if not lost then it may be repeated...
    num_rep = 0
    for _ in range(FINITE_DUPLICATION):
        if random.random() < G[source][dest]['repP']:
            num_rep += 1

    # Simulate delivery
    for _ in range(num_rep):
        deliver(G, dest, color)
```

Best-effort broadcast protocol executed by p

upon B-broadcast(m) do
  foreach $q \in \Pi$ do
    send m to q

upon receive(m) do
  B-deliver(m)

BEB side-notes

- Works with Perfect Links and No-Crash-Failure
- Doesn't work without the above assumptions!
- Message propagation/processing not modeled so far...

➢ let's advance the model for playing with R-Broadcast!!!

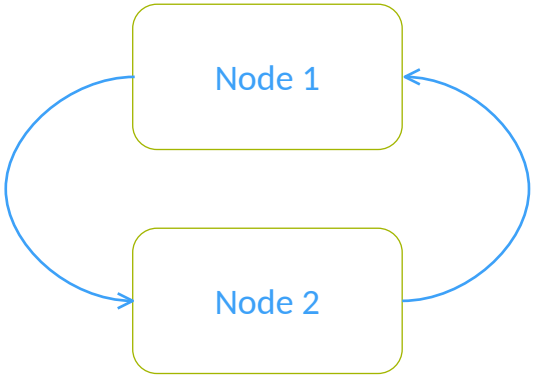
  ➢ We will do it thanks to Discrete Event Simulation (DES)

DES Ping Pong Example


```
on init:
  n1 = Node(), n2 = Node()
  n1.count = n2.count = 0
  scheduleEvent(<from: n1, to: n2, msg >, now + exp(1))

on event(event):
  node_from, node_to, msg = event

  // increment recipient counter
  node_to.count++
  // play ping pong, swap transmitter and receiver
  scheduleEvent(<node_to, node_from, msg>, now + exp(1))
```



DES Simulator Structure

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

Components and variables:

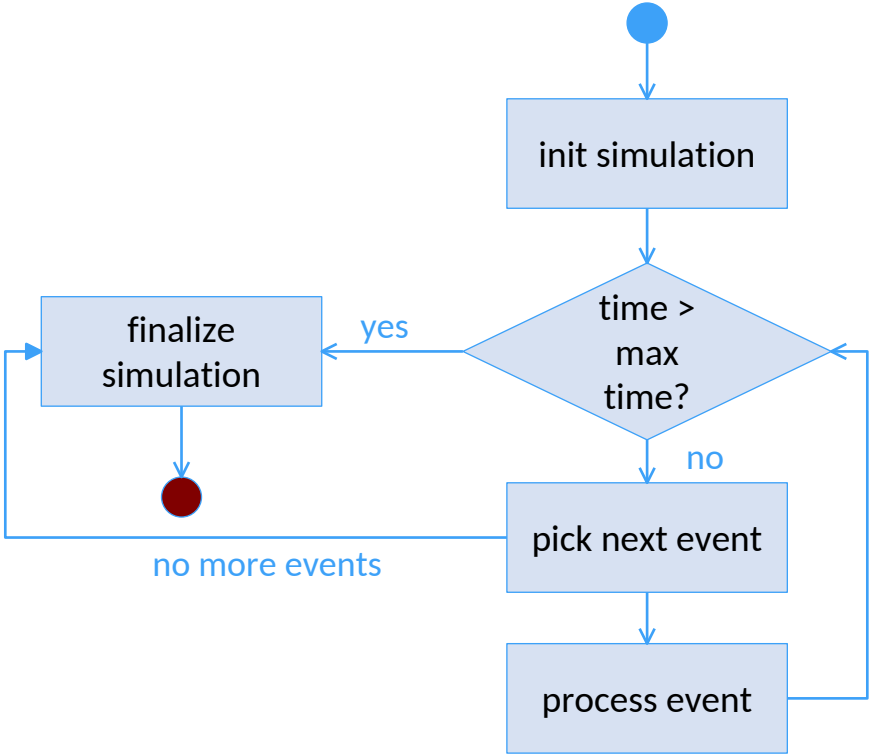
- An ordered queue (list) of events
- Current time
- Variables for performance monitoring (e.g., # of events)
- Modules implementing the behavior of system components (models)

Working principle:

- Initialize simulation modules
- Pick the first (in terms of time) event from the queue
- Update current time and check for terminating condition
- Invoke the event handling of the destination module
- Repeat

DES Simulator Structure

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

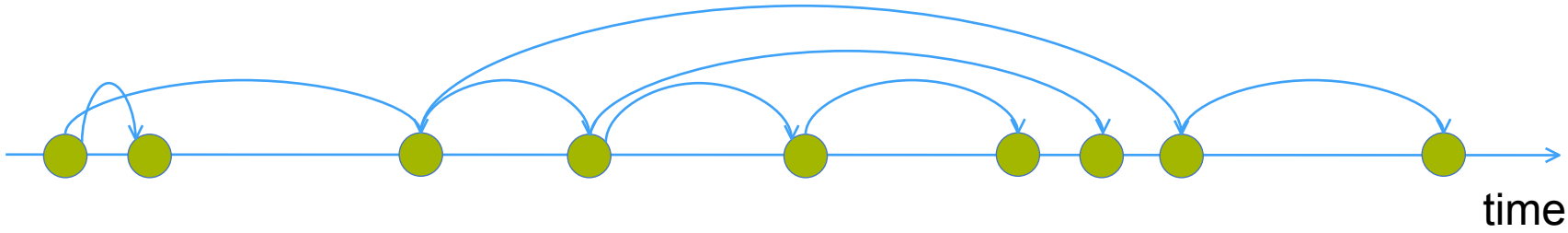


- New events are generated processing events
- Generated events are inserted with the proper (time) order in the event queue

- Minimal Python class to manage push/pop of events from an events queue will be provided :)

DES visualization

- event-driven: evolution by the generation and the consumption of events



R-Broadcast simulation

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

As for BEB we have to...

1. Create a complete graph
2. Define Node and Channel Failure

   PARAMETRIZED models
3. **R-Broadcast from pseudo to python!**

Once we have the above implemented...

1. Verify RB properties under "No Failure" and "With Failure" setting
2. Visualize RB under the two settings :)

Reliable broadcast protocol executed by p

upon initialization **do**
  SET $delivered \leftarrow \emptyset$ % Messages already delivered

upon R-broadcast(m) **do**
  send m **to** $\Pi - \{p\}$
  R-deliver(m)
  $delivered \leftarrow delivered \cup \{m\}$

upon receive(m) *from* q **do**
  if not $m \in delivered$ **then**
    send m **to** $\Pi - \{p, q\}$
    R-deliver(m)
    $delivered \leftarrow delivered \cup \{m\}$

Provided code before solutions...

- EventScheduler.py
  - class to manage events queue
  - schedule event ==> push event, after popping events they must be processed
- r_broacast.py
  - main file including the typical "while loop" for processing events
- MyProcessTemplate.py
  - DES class describing a Process
  - A Process should be able to r_broadcast and receive messages
  - try to complete the code that is missing :)

MyProcess.py schema

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

```
delivered = {}

r_broacast(msg, recipients):
  // logic of sending msg to recipients
  // SIMULATE SENDER CRASH!!!
  // logic of R-delivering
...

on_receive(src, msg):
   // logic of receving/processing msg sent by src
   // SIMULATE CHANNEL FAILURE!!!
...
```

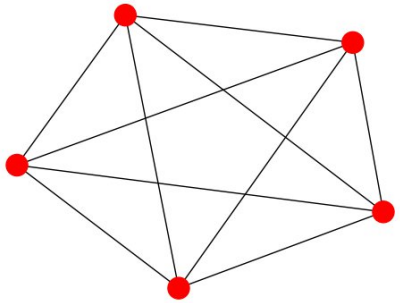
- A bunch of further class methods provided
- **NB:** If you complete the template as expected... logging events you should obtain as ouput a "Sequence Diagram Text File" that can be rendered through the seqdiag library
- Checkout this: http://blockdiag.com/en/seqdiag/introduction.html

to install and learn the seqdiag expected format

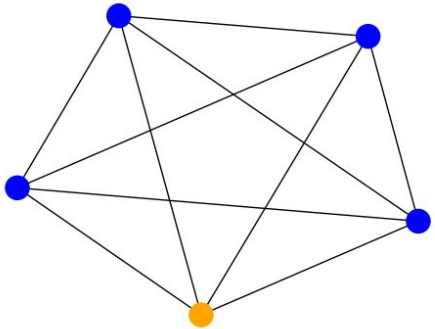
Wishful Output

UNIVERSITY OF TRENTO - Italy
Information Engineering
and Computer Science Department

BEFORE BROADCAST

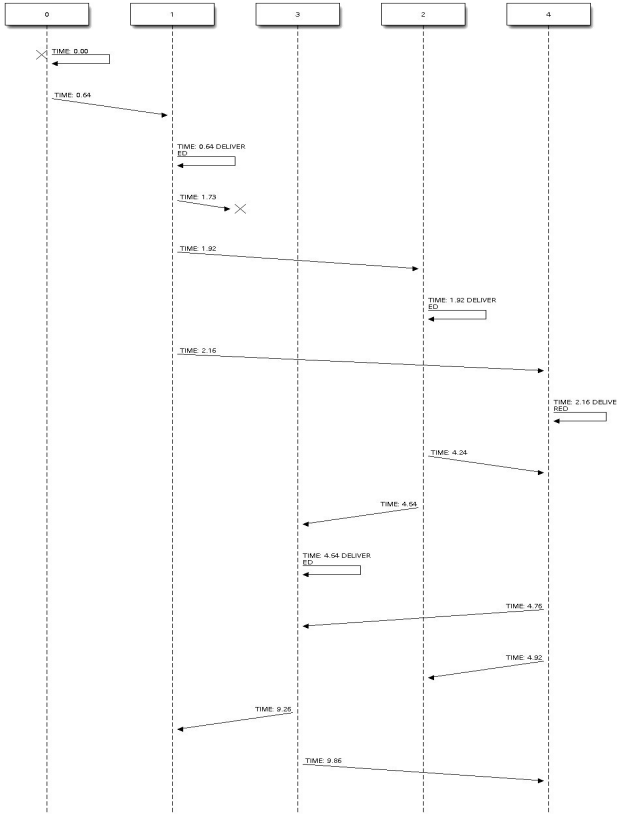


AFTER BROADCAST



Sequence Diagram

Tip: export to svg and open it in a browser



Questions?