

A Rendezvous in Network Experiment — Case Study of Kuroyuri

Ken-ichi Chinen
Internet Research Center,
JAIST
info@starbed.org



Outline

Goal: Automatically driving of network experiments

We developed...

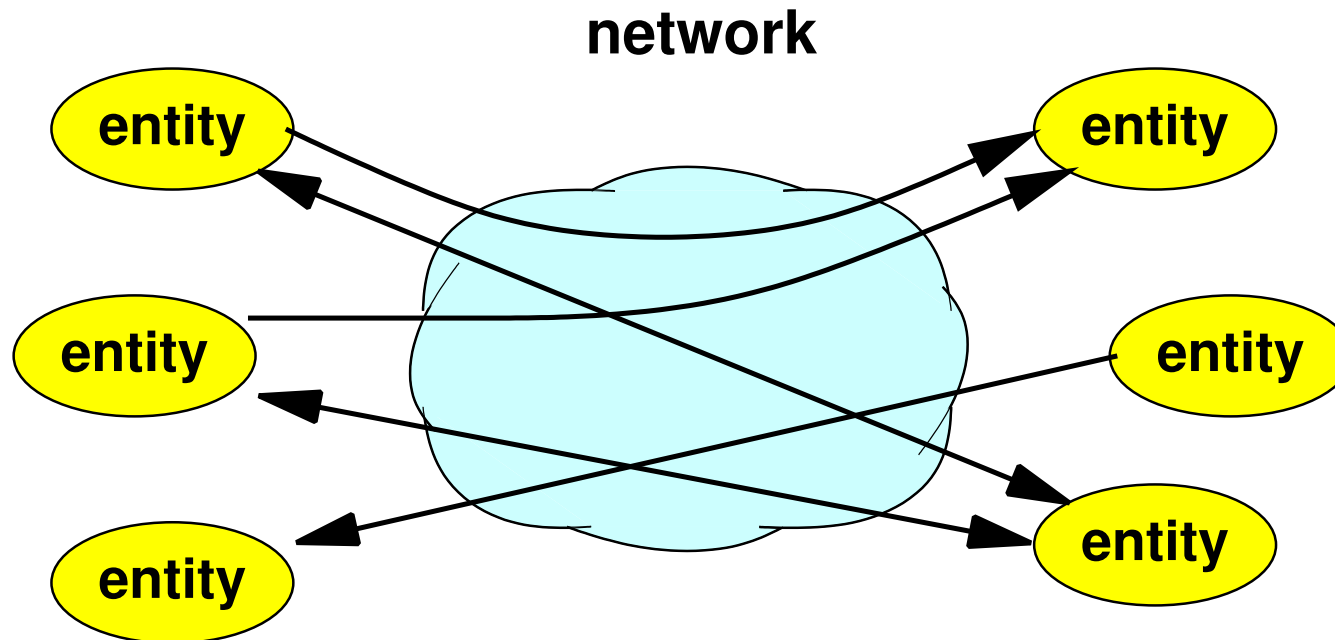
- Experiment description language
- Evaluator of the language
- Resource management program

This talk aims an introduction of **Rendezvous**

- It is important to control experiment steps

Network Experiment — Why ?

We are living with computer network.
Their activities have to be checked
like cars, air crafts and other industries.



Network Experiment

Experiment is ...

- Verification of program/service specification
 - ◇ Protocols
 - ◇ Data formats
 - ◇ Timing
- Evaluation of their performance
 - ◇ Thru-put
 - ◇ Response time

Requirements of Driving System

- Scalability
 - ◇ Supports many entities
- Handling program w/ no modification
 - ◇ Supports binary distributed program
Do you want modify Oracle or something ?
- Programmable & traceable
- Management the order of steps

⇒ We design the driving system, Kuroyuri.

Our Approach — Kuroyuri

It is designed as a language processing system

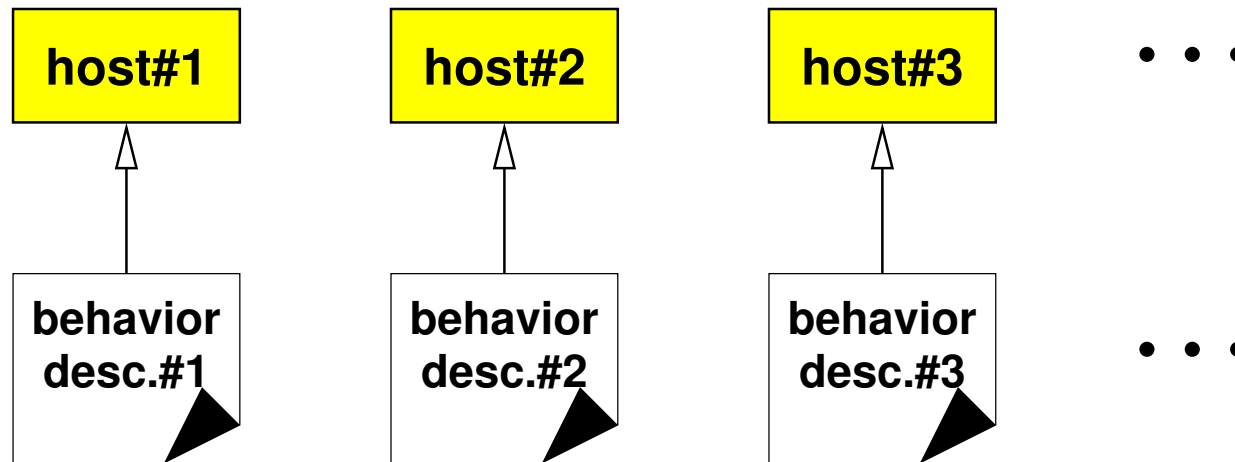


Distributed processing language

- Programmable & traceable
- Process experiment steps by calling of external programs
- Enables communication between entities
- Runs with single file

Traditional Approach

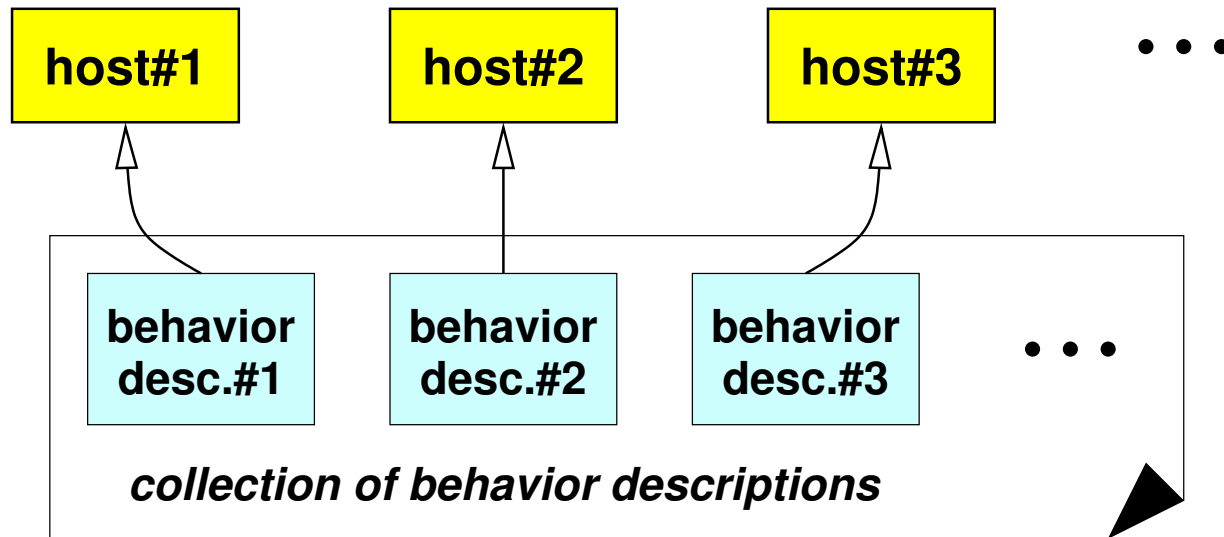
Single host-base scripting, requires a lot of files



Can you image the situation w/ 100+ hosts ?

- You may lose files
- Most of them is same or almost similar

(Ours) Script Archiving



- The program drives an experiment w/ single file
→ Easy handling
- **Class** absorbs difference of similar scripts
⇒ It reduces waste disk space

Language Design

Called "*Experiment Description Language*"

- User can describe many things
 - ◇ Node (NICs)
 - ◇ Network (IP address)
 - ◇ Topology — Binding of nodes and network
 - ◇ Scenario — Behavior of nodes
- User can write script easily
 - ◇ Light weight language

This presentation focuses **Scenario** .

Definition the Term "Scenario"

Scenario = a sequence of statements
= a set of statements with an order

Statement

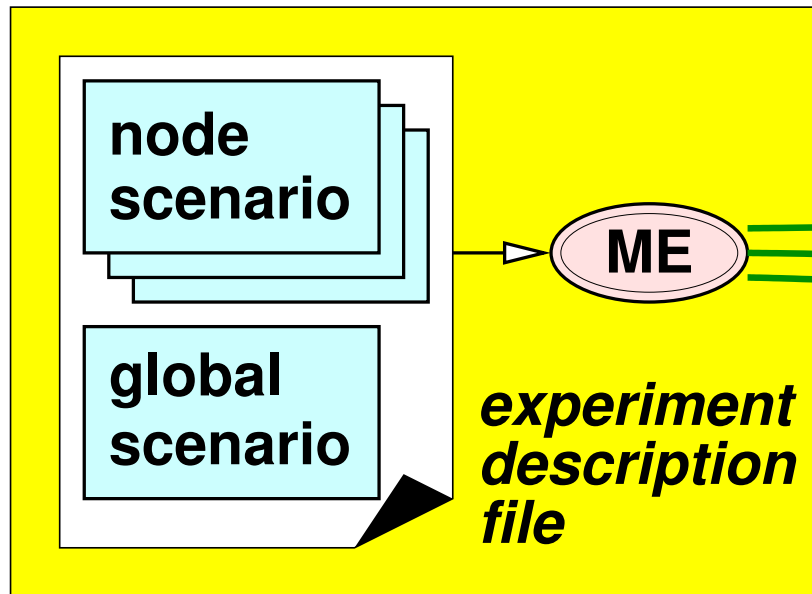
- Program wake-up/down
- Messaging
- Flow controls

Then, scenario forms...

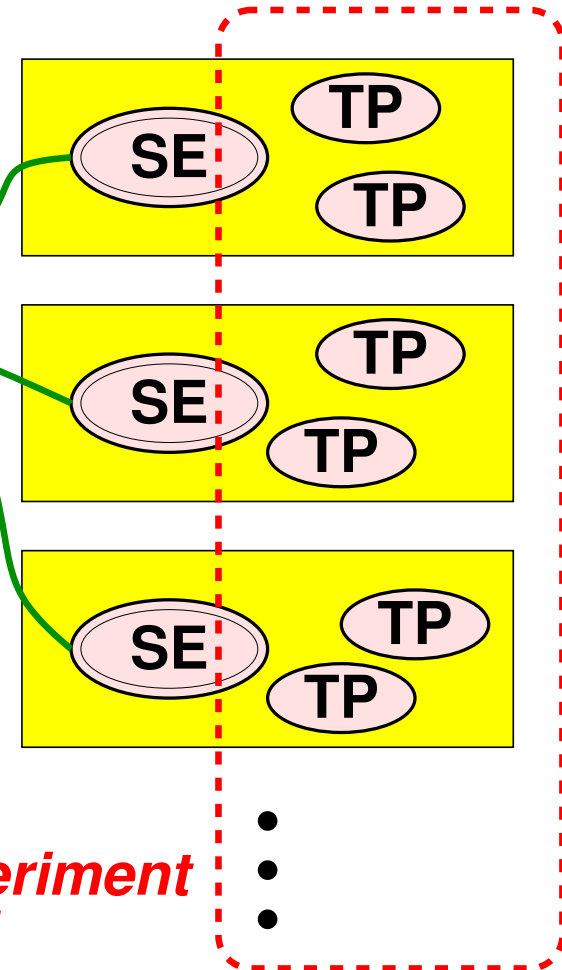
- Behavior of nodes (***node scenario***)
- Behavior of all nodes (***global scenario***)

driving host

to be driven hosts



connections
for control



ME master evaluator
SE slave evaluator
TP Target program

Fundamental Statements

wake call program

wakewait call program and wait its termination

kill terminate program

sleep sleep

send send message

recv receive message

if condition switch

for/while condition loop

Driving Experiment Step by Step

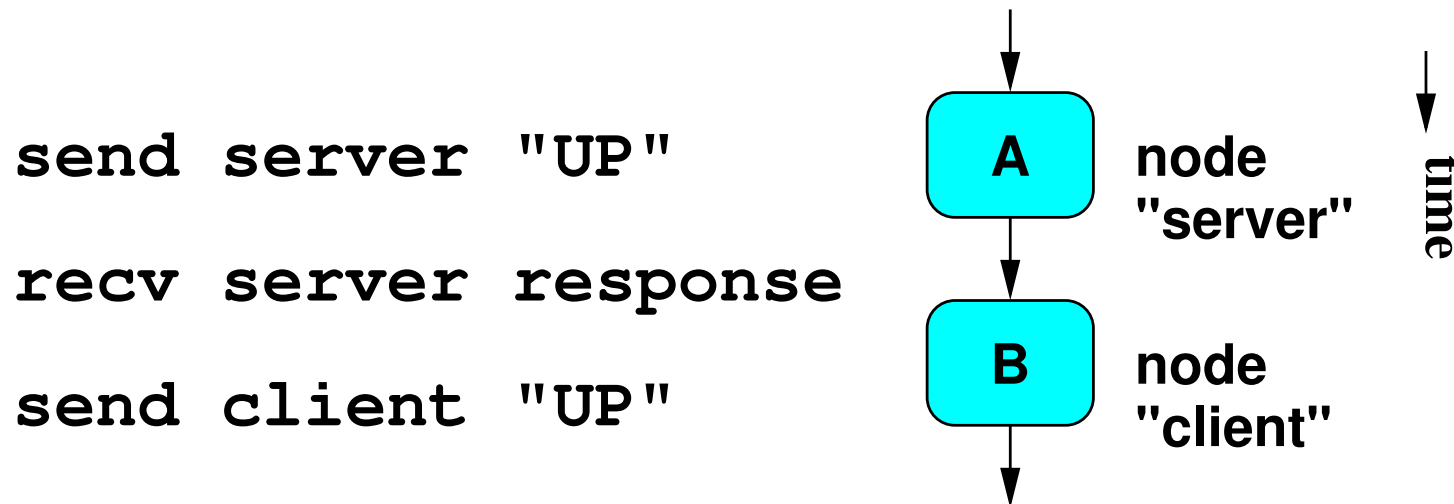
To manage the order of steps is one of most important things in network experiment.

- Satisfy dependencies
 - ◇ e.x. *Wake up client after server waked*
- Synchronization
 - ◇ Simultaneous execution

If evaluator waits remote's message as acknowledgment, it can drive step by step.

Serial Flow

To use `recv` makes a waiting of the remote message in serial flow.

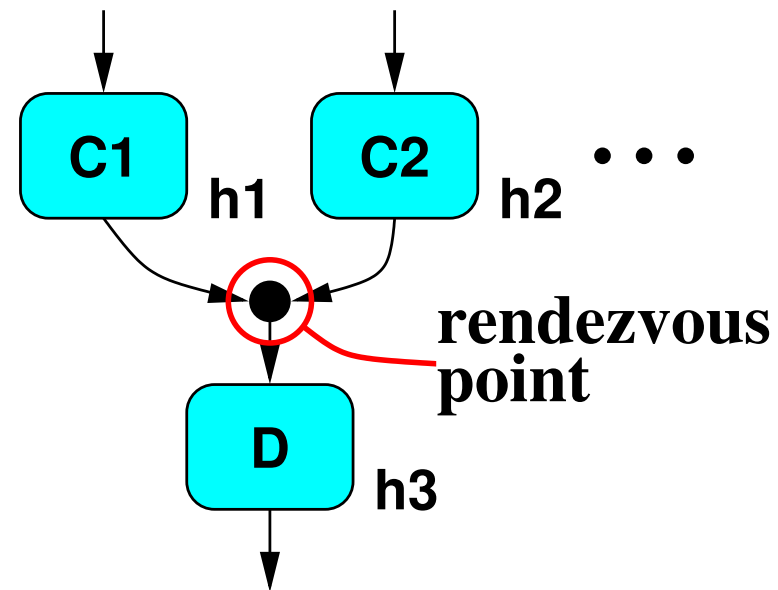


But it is not enough to drive parallel flow. Because it does not support multiple inputs on concurrent connections.

Parallel Flow

`sync` waits multiple messages.

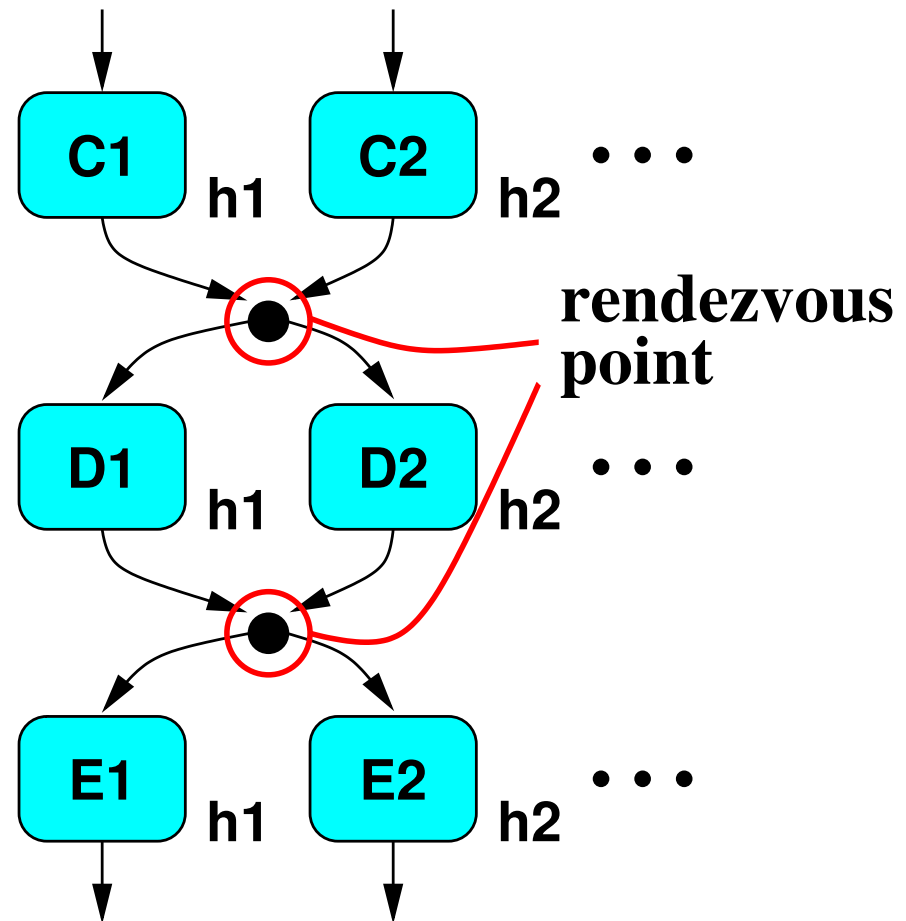
```
send h1 "C"  
send h2 "C"  
...  
sync {  
  msgmatch h1 "ready"  
  msgmatch h2 "ready"  
  ...  
}  
send h3 "D"
```



It makes a rendezvous point.

Simultaneous Execution

```
send h1 "C"  
send h2 "C"  
...  
sync {  
  msgmatch h1 "ready"  
  msgmatch h2 "ready"  
  ...  
}  
send h1 "D"  
send h2 "D"  
...  
sync {  
  msgmatch h1 "ready"  
  msgmatch h2 "ready"  
  ...  
}  
send h1 "E"  
send h2 "E"  
...
```



Sync — syntax

`sync` consists of conditions.

```
sync {  
  msgmatch foo "ready"  
  msgmatch bar "ok"  
}
```

Evaluator waits messages node-`foo`'s "ready" and node-`bar`'s "ok"

`sync` is atomic operation.
It waits until conditions are satisfied.

Sync — implementation

- Loop of condition evaluations
 - ◇ **msgmatch**
 - 1) Check connections using nonblocking I/O
Current version uses system-call `poll`
 - 2) Parse data in buffer as message
 - 3) Match message specified and gotten
 - ◇ **timeout**
 - ★ Give-up; break this loop
- Short time sleep to avoid heavily busy loop

Related Works

Shell, rsh, ssh and similar

- They are single host-base scripting
- They require a lot of files
 - ⇒ it may cause mistake and confusion

CORBA

- It is a strong framework for distributed computing
- However, it forces to according to its manner
 - ⇒ it is not suit to control binary distributed programs

Future Works

- Design for exceptions
`catch/throw` or other styles
- Scalability
We already used the system on 250+ hosts.
However, we should try more many hosts
to support cellar phones, sensors and others
- Consideration for collaboration with other distributed processing models/languages
 - ◇ Linda, druby

Summary

- Introduce rendezvous in Kuroyuri
 - ◇ Message-base rendezvous
 - ◇ Useful to describe...
 - ★ Dependency
 - ★ Simultaneous execution
- Scenarios are written in single file
 - ◇ To avoid operator's mistake and confusion

THANK YOU

Contact points

WWW <http://www.starbed.org/>
E-mail info@starbed.org

Keywords for search engines

StarBED project, SpringOS, Kuroyuri