CS 525: Advanced Topics in Distributed Systems Spring 2010

Indranil Gupta

Structuring Project Code:

"The 1 Line Solution"

Background

Discussion – Studying Your Protocol

- How accurate are mathematical analyses?
  - Often simplistic, so we resort to simulations, often trace-based…
- Simulations easy to do – implement, and run on your machine (or a small cluster)
- How accurately can simulations model real-world stresses?
- How do we know that we're accounting for all possible kinds of failure?
- All possible kinds of stresses? All possible kinds of traces?

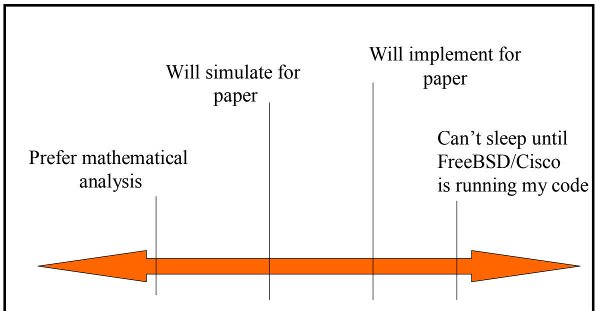
Discussion – Studying Your Protocol

- *Can* simulations ever model reality accurately?
- Is deployment the ultimate test?
- Have you seen any papers that match simulation and real-world running experimental numbers?
- Why?

- Unfortunately, often "The paper is the system" in research

As a Result

- Rare for someone else to pick up your idea, implement it and run it in the real world (although it does happen, there are too many ideas out there…)



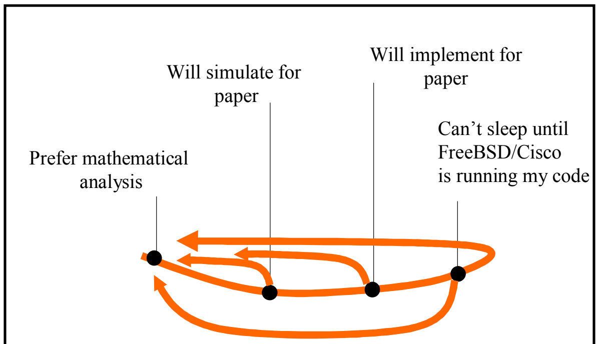
Prefer mathematical analysis

Will simulate for paper

Will implement for paper

Can't sleep until FreeBSD/Cisco is running my code

Research Spectrum



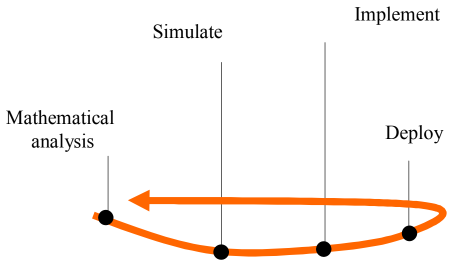
Will simulate for paper

Will implement for paper

Can’t sleep until FreeBSD/Cisco is running my code

Prefer mathematical analysis

Ideal (but difficult) Research Cycle



Simulate

Implement

Mathematical analysis

Deploy

Remember: Real World Code deployment is a cycle

Presumption

- Assumption: Rare for someone else to pick up your idea, implement it and run it in the real world (although it does happen, there are too many ideas out there…)
- Deployment is mostly your responsibility
- Problem: Design your simulation code so that you can convert your code from simulation → deployable version by changing a single line of code
- “1 Line Solution”

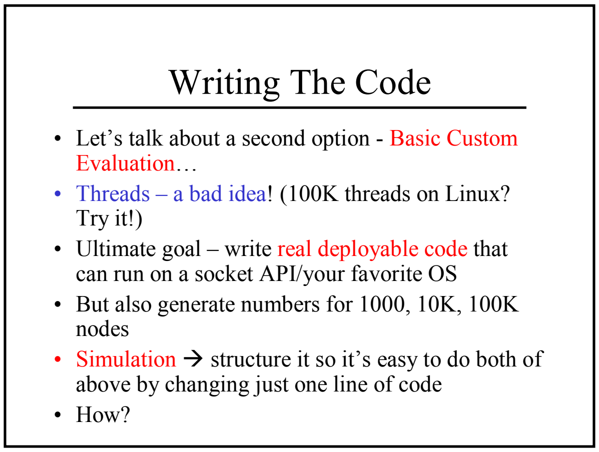
For Your Project

“How do I write code for my Distributed Protocol XYZ so that I can evaluate it with 100, 000 nodes?”

The 1 Line Solution

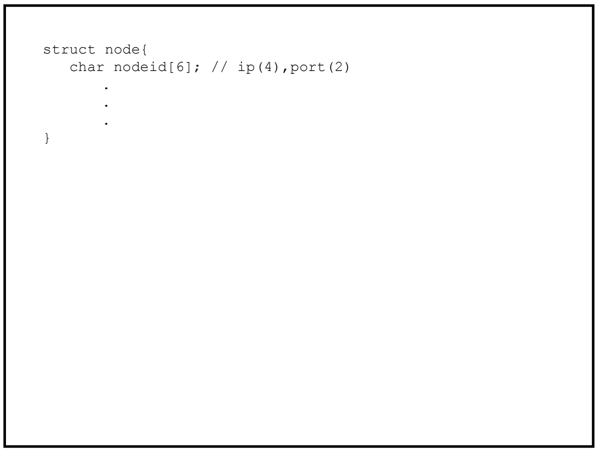
Writing The Code

- Simulation engines (ns2, glomosim) etc. are one option
- A required standard in some research communities (e.g., ad-hoc networking)
- Not so in the p2p or (largely) the sensor net communities (yet)

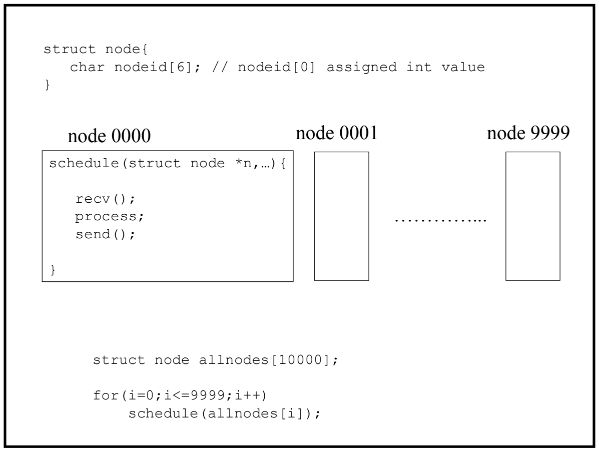


Writing The Code

- Let's talk about a second option - Basic Custom Evaluation…
- Threads – a bad idea! (100K threads on Linux? Try it!)
- Ultimate goal – write real deployable code that can run on a socket API/your favorite OS
- But also generate numbers for 1000, 10K, 100K nodes
- Simulation → structure it so it's easy to do both of above by changing just one line of code
- How?



```
struct node{
   char nodeid[6]; // ip(4),port(2)
          .
          .
          .
}
```



```
struct node{
   char nodeid[6]; // nodeid[0] assigned int value
}
```

node 0000

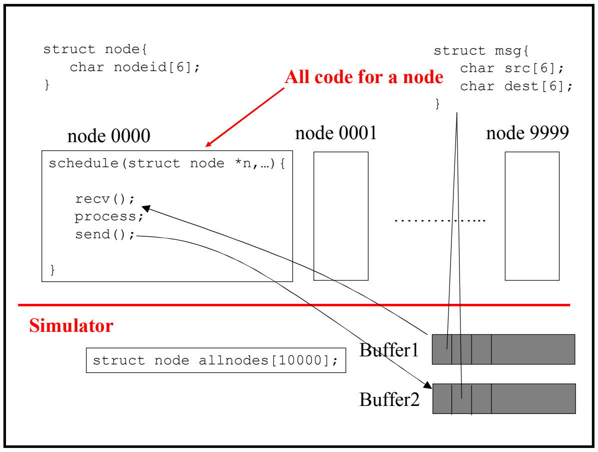
```
schedule(struct node *n,…){

   recv();
   process;
   send();

}
```

node 0001

……………..

node 9999

```
struct node allnodes[10000];

for(i=0;i<=9999;i++)
    schedule(allnodes[i]);
```



```
struct node{
   char nodeid[6];
}
```

```
struct msg{
   char src[6];
   char dest[6];
}
```

All code for a node

node 0000

```
schedule(struct node *n,…){

   recv();
   process;
   send();

}
```

node 0001

……………..

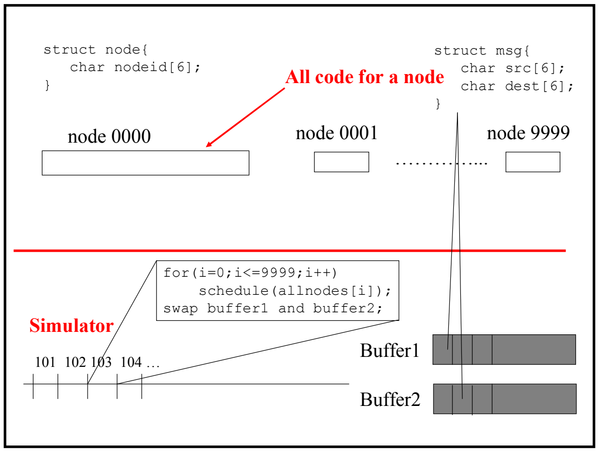
node 9999

Simulator

```
struct node allnodes[10000];
```

Buffer1

Buffer2



```
struct node{
   char nodeid[6];
}
```

```
struct msg{
   char src[6];
   char dest[6];
}
```

All code for a node

node 0000

node 0001

……………..

node 9999

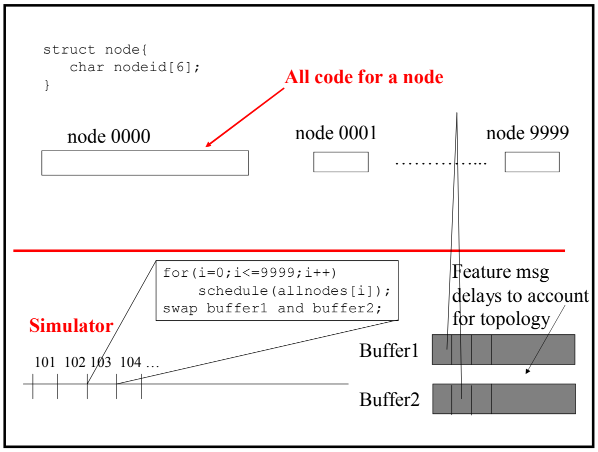
```
for(i=0;i<=9999;i++)
    schedule(allnodes[i]);
swap buffer1 and buffer2;
```

Simulator

101 102 103 104 …

Buffer1

Buffer2



```
struct node{
   char nodeid[6];
}
```

All code for a node

node 0000

node 0001

……………..

node 9999

```
for(i=0;i<=9999;i++)
    schedule(allnodes[i]);
swap buffer1 and buffer2;
```

Feature msg delays to account for topology

Simulator

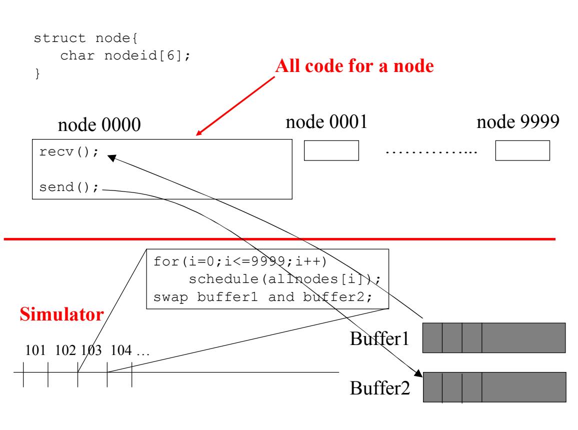
101 102 103 104 …

Buffer1

Buffer2

The advantage of such an elaborate spread?

- Layering gives clean separation of implementation from simulation
- Easy debugging (No global variables for the implementation, please!)
- And…



```
struct node{
   char nodeid[6];
}
```

All code for a node

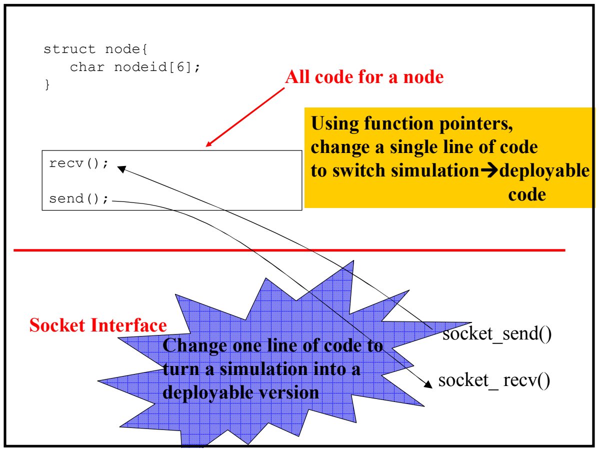
node 0000 node 0001 node 9999

```
recv();

send();
```

…………..

```
for(i=0;i<=9999;i++)
    schedule(allnodes[i]);
swap buffer1 and buffer2;
```

Simulator



```
struct node{
   char nodeid[6];
}
```

All code for a node

Using function pointers, change a single line of code to switch simulation→deployable code

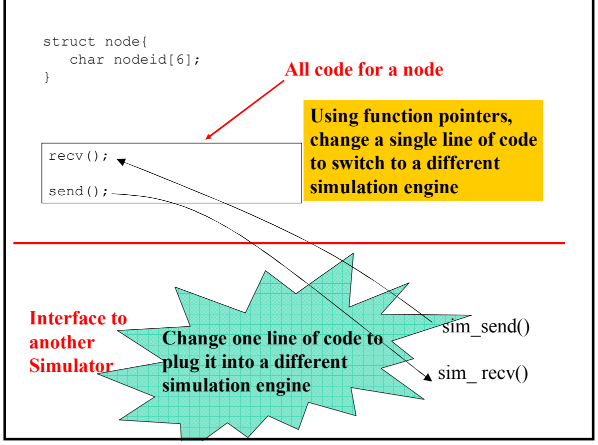
```
recv();

send();
```

Socket Interface

Change one line of code to turn a simulation into a deployable version

socket_send()

socket_ recv()



```
struct node{
   char nodeid[6];
}
```

All code for a node

Using function pointers, change a single line of code to switch to a different simulation engine

```
recv();

send();
```

Interface to another Simulator

Change one line of code to plug it into a different simulation engine

sim_send()

sim_ recv()

- Easier to do above with C or Java or C++
- Can put an "Application" layer on top of the "Real Code" layer
- Of course, you are free to structure your code in a different way should you so wish…

Questions