

2nd Mini-Project: Routing Simulation

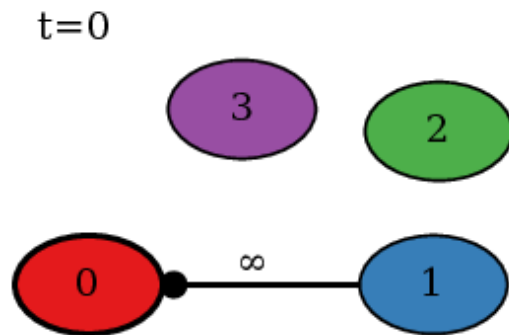
Routing Protocols

Overview

What you'll learn:

- Routing Protocols
 - Link State
 - Distance Vector (w/ & w/o RPP)
 - Path Vector
- Event Based Simulation

Implement Routing Protocols
in a Simulated Network

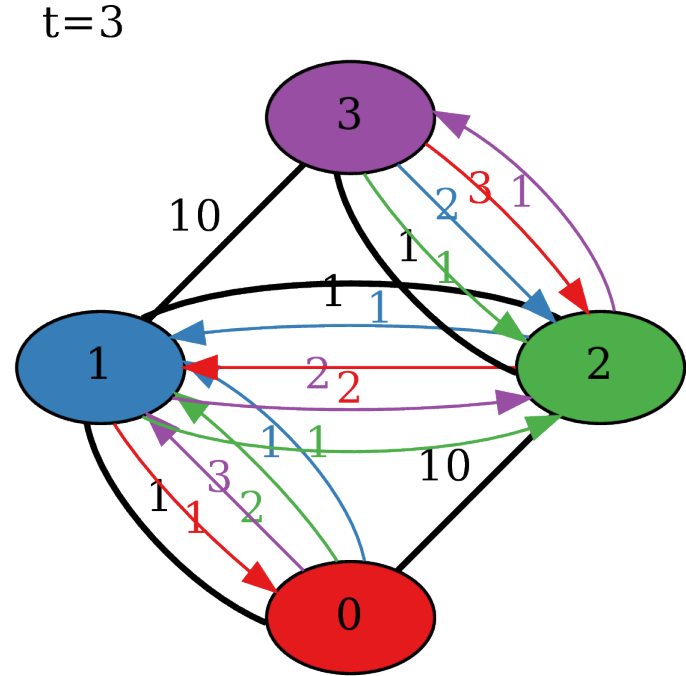


Implementing a Routing Protocol

Implement Handlers

```
void notify_link_change(  
    node_t neighbor,  
    cost_t new_cost);
```

```
void notify_receive_message(  
    node_t sender,  
    void *message);
```



Implementing a Routing Protocol

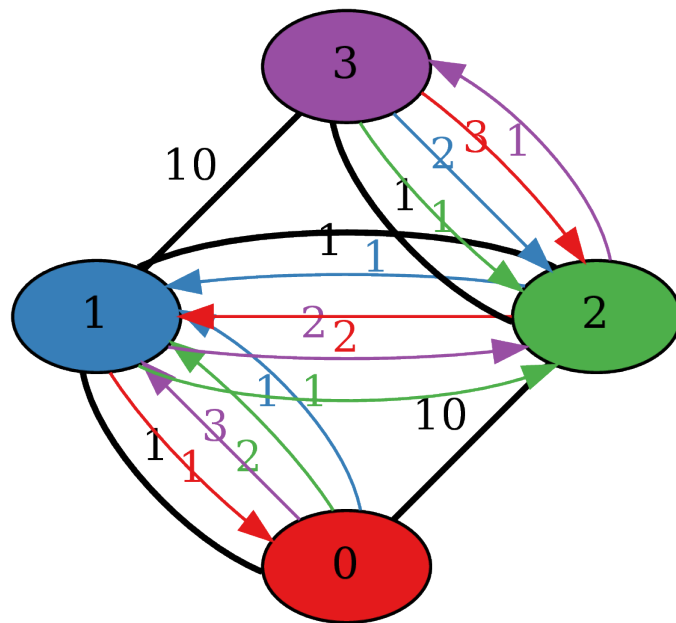
Use API

Topology:

```
node_t get_current_node();
node_t get_first_node();
node_t get_last_node();
cost_t get_link_cost(
    node_t neighbor);
cost_t COST_ADD(cost_t a, cost_t b)

for (node_t n = get_first_node();
     n <= get_last_node(); n++) {
    if (n != get_current_node()) {
        ...
    }
}
```

t=3

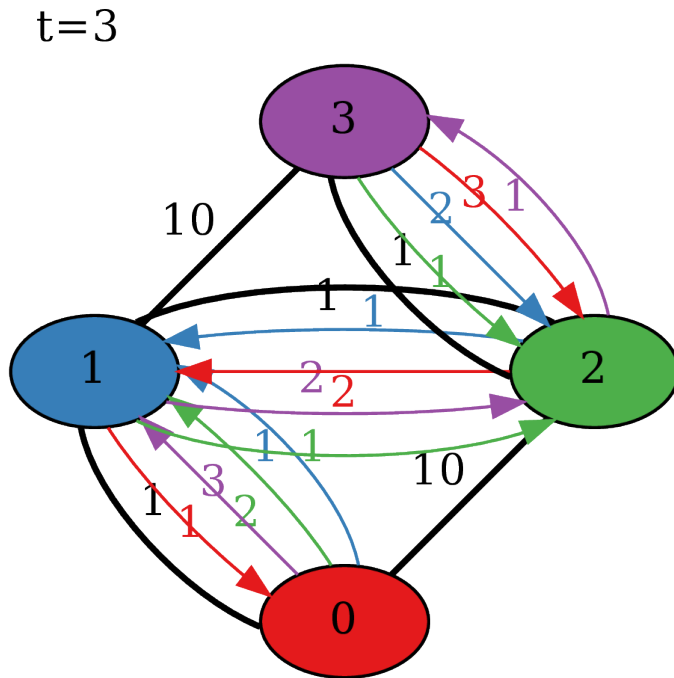


Implementing a Routing Protocol

Use API

Data Plane:

```
void set_route(  
    node_t destination,  
    node_t next_hop,  
    cost_t cost);
```



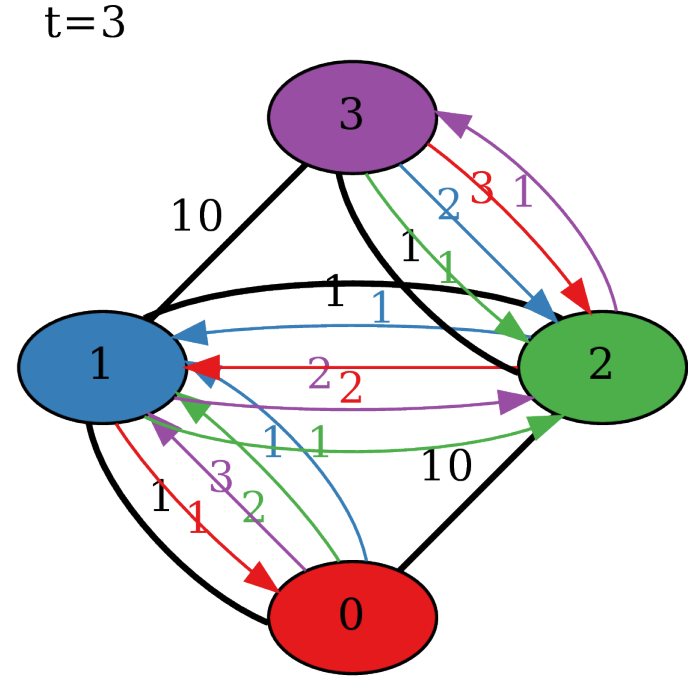
Implementing a Routing Protocol

Use API

Messages :

```
void send_message(  
    node_t neighbor,  
    void *message);
```

```
msg_t *m  
    = malloc(sizeof(msg_t));  
m->data = ...;  
send_message(1, m);
```



Implementing a Routing Protocol

Use API

State:

No Global/Static Variables!

```
void *get_state();
```

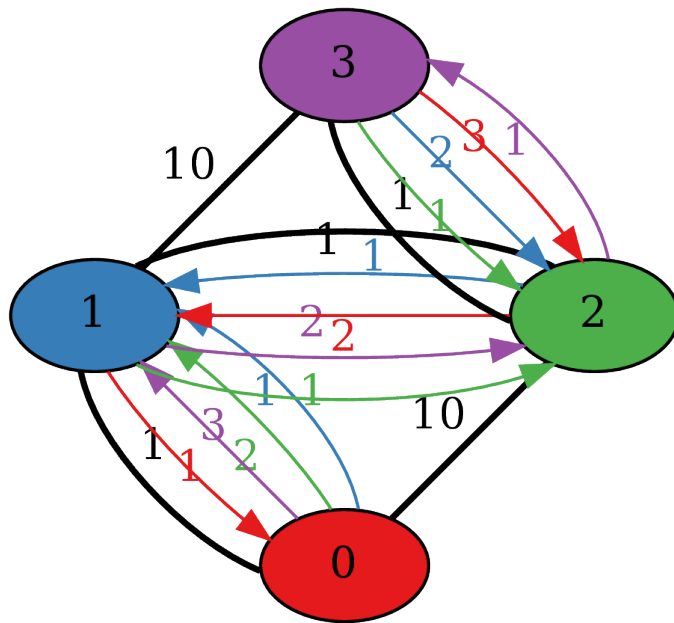
```
void set_state(void *state);
```

```
state_t *s;
```

```
if (!(s = get_state())) {  
    s = malloc(sizeof(state_t));  
    set_state(s);  
}
```

```
s->data = ...;
```

t=3



Four Routing Protocols

Distance Vector:

- Nodes exchange cost to each destination
- Bellman-Ford algorithm
- Send updates when changed

Distance Vector w/

Reverse Path Poisoning:

- Same, but set reverse path = ∞

Path Vector:

- Like DV, but send whole route instead of cost

Link State:

- Nodes share topology
- Dijkstra's algorithm

Link State: Sharing Topology

```
typedef struct {  
    cost_t link_cost[MAX_NODES];  
    int version;  
} link_state_t;
```

```
typedef struct {  
    link_state_t ls[MAX_NODES];  
} message_t;
```

Initialization:

- Local = get_link_cost();
Other = ∞ ;
version = 0
- Send to neighbors

Link Change:

- Update local costs; version++
- Send to neighbors

Receive Message:

- Check for any newer version
- Update and send to neighbors

Running the Simulator

```
:~$ make
```

```
:~$ ls
```

```
dv-simulator  dvrpp-simulator
```

```
pv-simulator  ls-simulator
```

```
:~$ ./dv-simulator \  
      --steps-dot output.dot \  
      topology.net
```

```
:~$ ./dot-to-pdf.sh output.dot \  
      output.pdf
```

Four Simulators:

- dv-simulator
- dvrpp-simulator
- pv-simulator
- ls-simulator

Running the Simulator

```
:~$ make
```

```
:~$ ls
```

```
dv-simulator  dvrpp-simulator
```

```
pv-simulator  ls-simulator
```

```
:~$ ./dv-simulator \  
      --steps-dot output.dot \  
      topology.net
```

```
:~$ ./dot-to-pdf.sh output.dot \  
      output.pdf
```

Input Topology:

- Text File

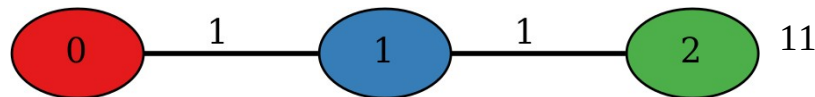
- Each line a link change:

time nodeA nodeB cost

- Bring links up one by one
- Change over time

- e.g. linear-3.net:

```
0 0 1 1  
0 1 2 1
```



Running the Simulator

```
:~$ make
```

```
:~$ ls
```

```
dv-simulator  dvrpp-simulator  
pv-simulator  ls-simulator
```

```
:~$ ./dv-simulator \  
      --steps-dot output.dot \  
      topology.net
```

```
:~$ ./dot-to-pdf.sh output.dot \  
      output.pdf
```

Output:

- DOT File, convert to PDF
- TMI, Filter:
 - epoch-steps
 - final-dot <dot-file>
 - hide-future-messages
 - show-routes-for <node>
 - steps-dot <dot-file>

Event Based Simulation

- Event Queue
 - Sorted by event time
 - Initialized with link changes from topology file
- Event Loop
 - Remove top event, process it
 - Add new events
 - Messages delivered next time epoch
 - Run until none left
- Simulation State
 - Dumped periodically
 - Each event, epoch, final
 - Outputted as pretty graph
- Caveats
 - No unnecessary messages
 - Will run forever
 - Can generate a lot of events
 - Limit to debug:
`--max-events <limit>`

Submission

- Develop your code on:
<https://git.rnl.tecnico.ulisboa.pt/>
- Implement: dv.c, dvrpp.c, pv.c, ls.c
- Do not change other files
- No build artifacts
- Tag submission as
project2-submission:
:~\$ git tag project2-submission
:~\$ git push origin project2-submission

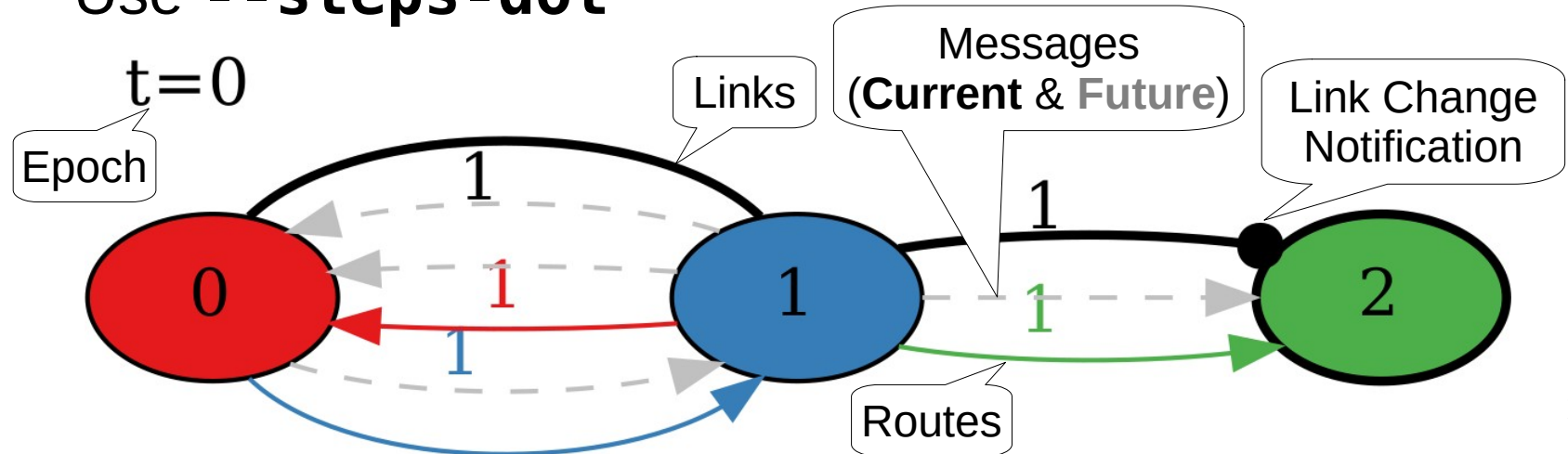
```
:~$ git clone <repo URL> .  
:~$ git checkout project2-submission  
:~$ ls  
routing-simulator.cpp Makefile dvrpp.c  
routing-simulator.h ls.c dv.c pv.c  
:~$ make  
:~$ ls  
(...)  
dv-simulator dvrpp-simulator  
pv-simulator ls-simulator
```

Nightly Builds

- Coming soon to a repo near you...
- Runs nightly
 - Simple tests – does not preclude running your own
 - Runs on master branch and generates build-report.md
 - Don't forget to pull
 - On request: must delete and push to rerun next time

Advice: Debugging

- Standard output/error will be ignored during grading
 - `printf("Node %d: ...", get_current_node(), ...)`
- Use **--steps-dot**



Improv

- Routing Protocol
 - DV, DVRPP, PV, LS
- Topology
 - linear-3
 - diamond
 - count-to-infinity
 - custom

