



Carnegie Mellon University

18-441/741 Spring 2021

Project 2 (Recitation 2/2)

Junbo Zhang

Mar 5

Make sure you have the newest version handout!

Project 2 Breakdown

Items	Points (100 – 441 + 110 – 741)
Setup (submit by checkpoint-1 deadline) <ul style="list-style-type: none"> - Parse configuration file - Generate UUID (uuid) and save it to config file - Add neighbor (addneighbor) - Node kill (kill) - Preliminary Design Document (1 page) - Successful compilation 	30 (Checkpoint 1) March 7 checkpoint: <ul style="list-style-type: none"> • Basic file operations • Get familiar with <u>socket</u> and <u>thread</u> programming
Peer Routing (submit by final deadline) <ul style="list-style-type: none"> - Reachability (neighbors) - Link state advertisement (map) - Priority Rank (rank) - Successful submission & compilation - Final Design Document (2-3 pages) 	70 (Final) March 28 submission: <ul style="list-style-type: none"> • Keep-alive message • Link state algorithm
Required for 18-741 (bonus for 441) Active distance metric	10 (Final) <ul style="list-style-type: none"> • Active distance metric

Recitation 1/2

Recitation 2/2

Start Early!

Display UUID (“uuid” command)

Keyboard Input:

```
uuid<newline>
```

Response: the uuid of the current node, e.g:

```
{"uuid": "f94fc272-5611-4a61-8b27-de7fe233797f"}
```

An example output for “uuid” command

Make sure it is implemented.

Note on Grading

Please make sure:

- Your functions for the checkpoint still work as required in your final submission.
- You don't delete any of the features you did for the checkpointing.

Failure on that might result in a deduction in your final score.

Note on Example Codes

Example codes are given to:

- provide an example of how to call socket APIs and how to send/receive messages.
- help you get a flavor of the communication required in this project.

It is NOT required that you use any part of them (you are encouraged to refer to them, though).

Content

1. Functions to implement for the final submission

- Make sure you read the handout carefully for full details

2. Keep-alive Message

3. Review on Link State Routing

4. Active Distance Metric *

Reachability (“neighbors” command)

Keyboard input:

neighbors<newline>

As expected, this output should change along with adding/killing nodes.

Response: a list of objects representing all active neighbors

```
[ {"uuid": "24f22a83-16f4-4bd5-af63-9b5c6e979dbb", "name": "node2", "host": "pi.ece.cmu.edu",  
  "backend_port": 18346, "metric": 10}, {"uuid": "3d2f4e34-6d21-4dda-aa78-796e3507903c",  
  "name": "node3", "host": "mu.ece.cmu.edu", "backend_port": 18346, "metric": 20} ]
```

An example output for “neighbors” command

The command “neighbors” should trigger an output of all active neighbors’ info.

How to distinguish between an “active” peer and an “inactive” one? Later.

Please read the handout for a comprehensive understanding.

Link State Advertisement (“map” command)

Keyboard input:

map

Response: An object representing an adjacency list for the latest network map. It should contain only **active** node/link. The object’s field name should be node’s name (by default), or UUID (if a node name was not specified.)

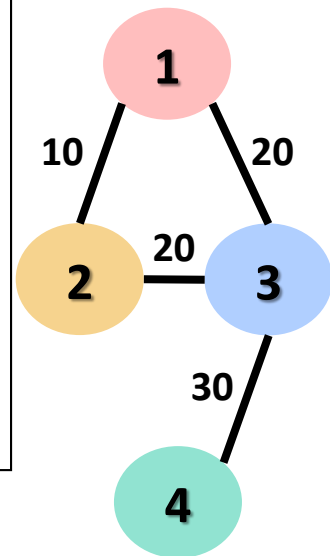
For example, this response should be generated for graph in figure 1 (assuming all nodes are active).

```
{ "node1":{"node2":10,"node3":20},  
  "node2":{"node1":10,"node3":20},  
  "node3":{"node1":20,"node2":20,"node4":30},  
  "node4":{"node3":30} }
```

As expected, this output should change along with adding/killing nodes.

An example output for “map” command

Please read the handout for a comprehensive understanding.



Priority Rank (“rank” command)

Keyboard Input

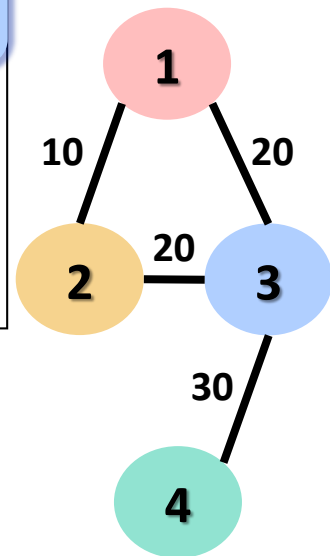
rank

Response: an ordered list (sorted by distance metrics) showing the distance between the requested nodes.

For example, the following response will be generated if was made to node1 (in figure 1).

```
[{"node2":10}, {"node3":20}, {"node4":50}]
```

As expected, this output should change along with adding/killing nodes.



An example output for “rank” command

Please read the handout for a comprehensive understanding.

Q & A

You are also welcomed to ask questions later (after chewing on the handout) in office hours.

Don't wait until deadlines; any of the TAs can answer.

Make sure you have the newest version handout!

Content

1. Functions to implement for the final submission 

2. Keep-alive Message

3. Review on Link State Routing

4. Active Distance Metric *

Keep-alive Message

For a node A with a neighbor B:

- A expects to periodically hear from B to ensure **B is reachable**
- A wants to tell B periodically that it is still **reachable for B**

You decide the specific implementation details.

- **Content** in the keep-alive message, **time gap** between consecutive messages, **threshold** on the number of missing messages, etc.

An example in the handout (not enforced; only suggestion):

- 1 keep-alive msg every 10 seconds
- Missing 3 consecutive msgs → Declare the peer node as unreachable

Some kind of sequence number might be needed (not enforced).



Q & A

You are also welcomed to ask questions later (after chewing on the handout) in office hours.

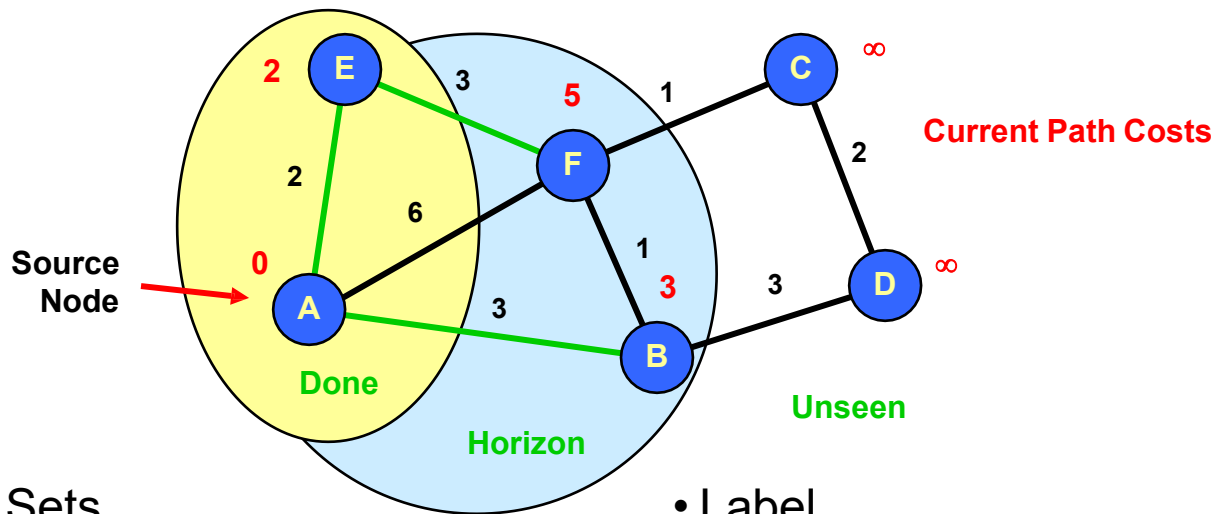
Don't wait until deadlines; any of the TAs can answer.

Make sure you have the newest version handout!

Content

1. Functions to implement for the final submission 
2. Keep-alive Message 
3. **Review on Link State Routing**
4. Active Distance Metric *

Dijkstra's Algorithm



• Node Sets

- Done
 - Already have least cost path to it
- Horizon:
 - Reachable in 1 hop from node in Done
- Unseen:
 - Cannot reach directly from node in Done

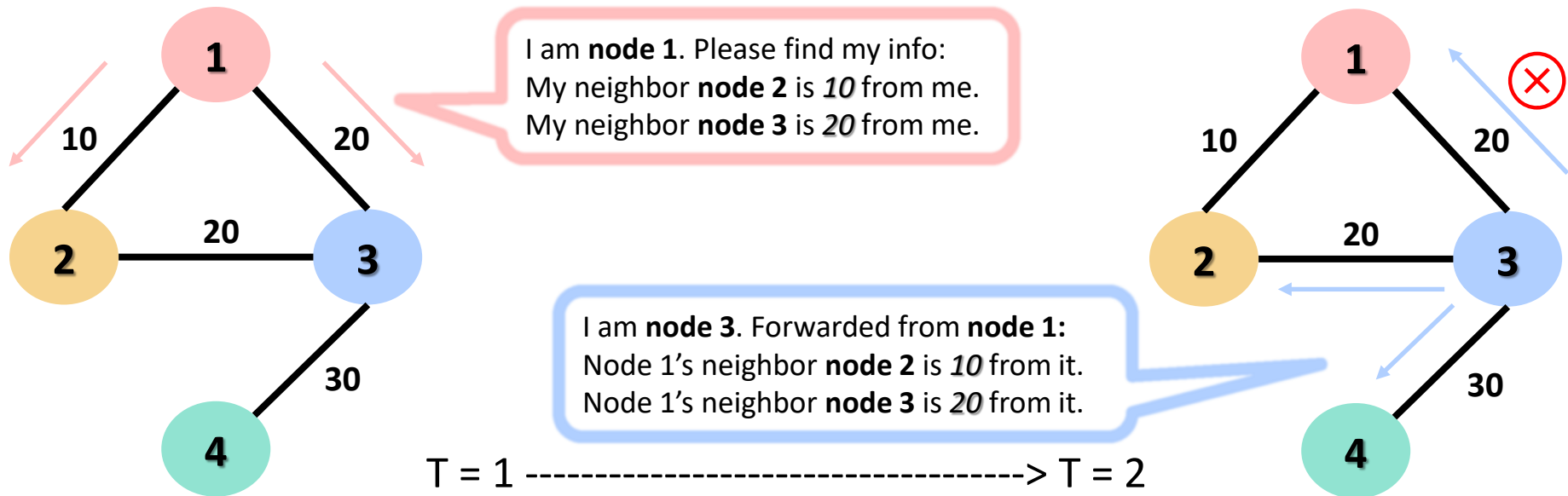
• Label

- $d(v)$ = path cost from s to v

• Path

- Keep track of last link in path

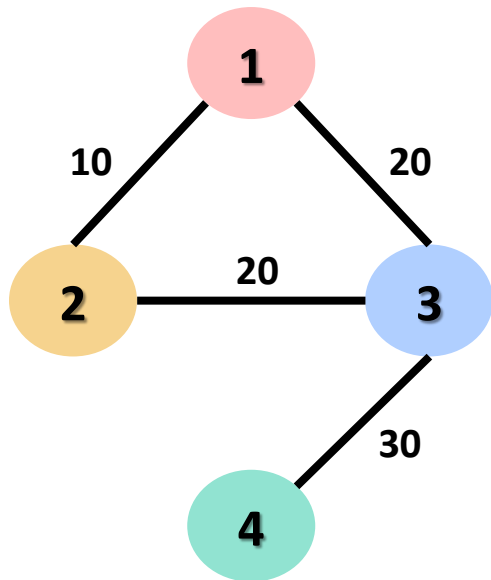
Link State Routing Review



Every node floods its neighbors with:

- Its link state advertisement message containing its neighbors' info
- Forwarded link state advertisement message from other nodes

Link State Routing Review



Upon receiving messages (originated/forwarded) from neighbors, each node:

- re-constructs the whole graph **independently**
- updates its graph if **new (different)** info received
- runs Dijkstra **independently** with itself as the source node

Eventually, each node:

- has the **whole** graph.
- knows the shortest path to every **other** node.

Difference Between Keep-alive and Link State?

Purely for this project, these two kinds of messages differ from each other on:

Keep-alive	Link State Advertisement
Periodic, more often	Triggered, less often*
Only to neighbors	Forwarding is required
Simple and short	More informative

* Link State Advertisement can also be periodic

You decide some potential “messy” details, in a way you think makes sense:

- Shall I flood a neighbor marked as unreachable?
- What is the format of a keep-alive/link state message?
- Receiving a forwarded message that was sent by myself originally?
- Port collision?
- ...




Q & A

You are also welcomed to ask questions later (after chewing on the handout) in office hours.

Don't wait until deadlines; any of the TAs can answer.

Make sure you have the newest version handout!

Content

1. Functions to implement for the final submission 
2. Keep-alive Message 
3. Review on Link State Routing 
4. **Active Distance Metric ***
 - Extra credit: 18-441. Required: 18-741.

Active Distance Metric

Your program is expected to activate this mode by setting “active_metric = 1” in “.conf”.

- By default (e.g., no lines regarding active_metric), it is inactive.

In one sentence: Distances between nodes can change over time.

You decide the specific implementation details.

- What it models. E.g., RTT (of keep-alive msgs) between nodes (not enforced).
- Frequency of change. E.g., every 10 seconds (not enforced).
- Implemented; works; makes sense; you are good.

Q & A

You are also welcomed to ask questions later (after chewing on the handout) in office hours.

Don't wait until deadlines; any of the TAs can answer.

Make sure you have the newest version handout!