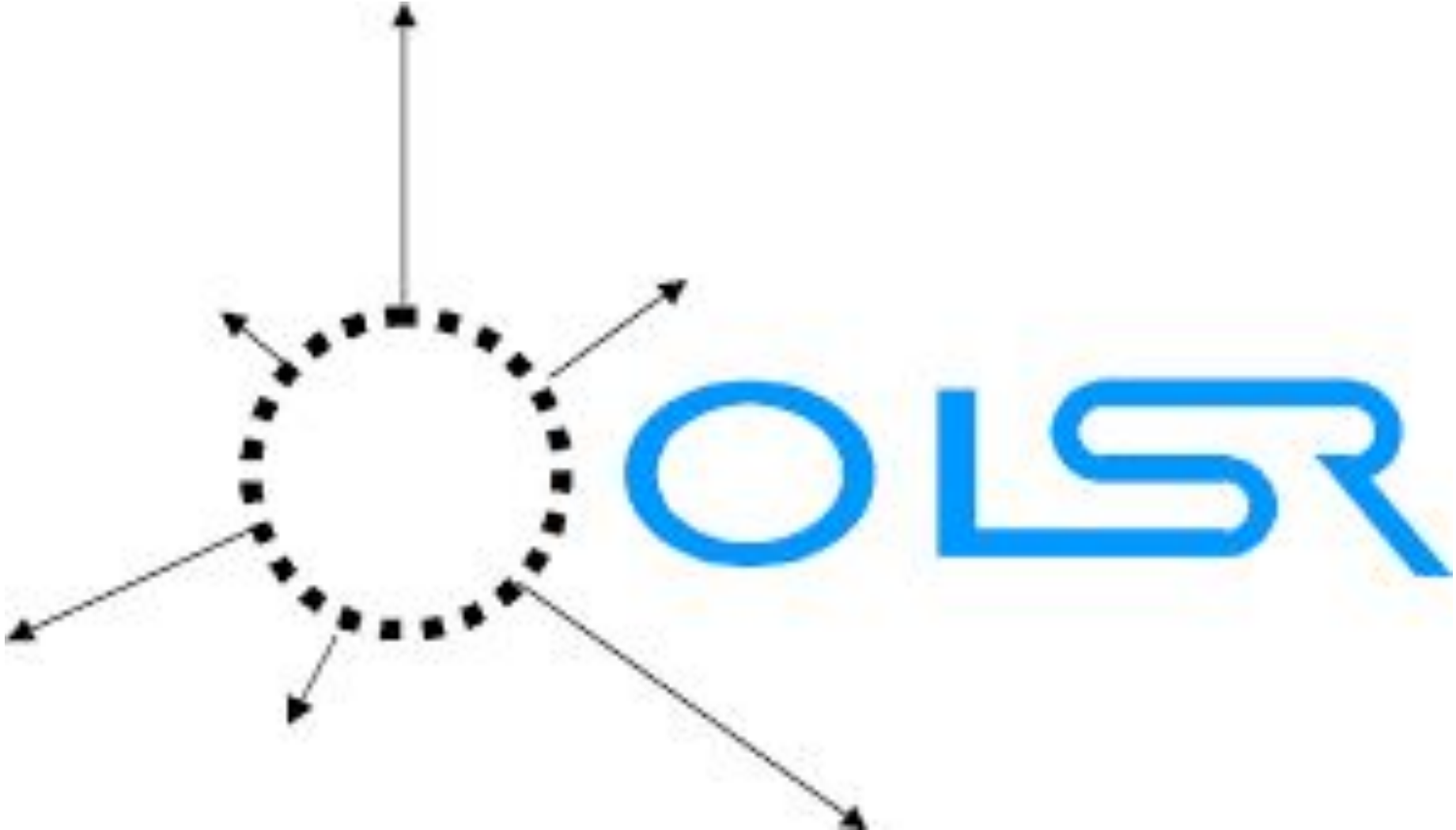
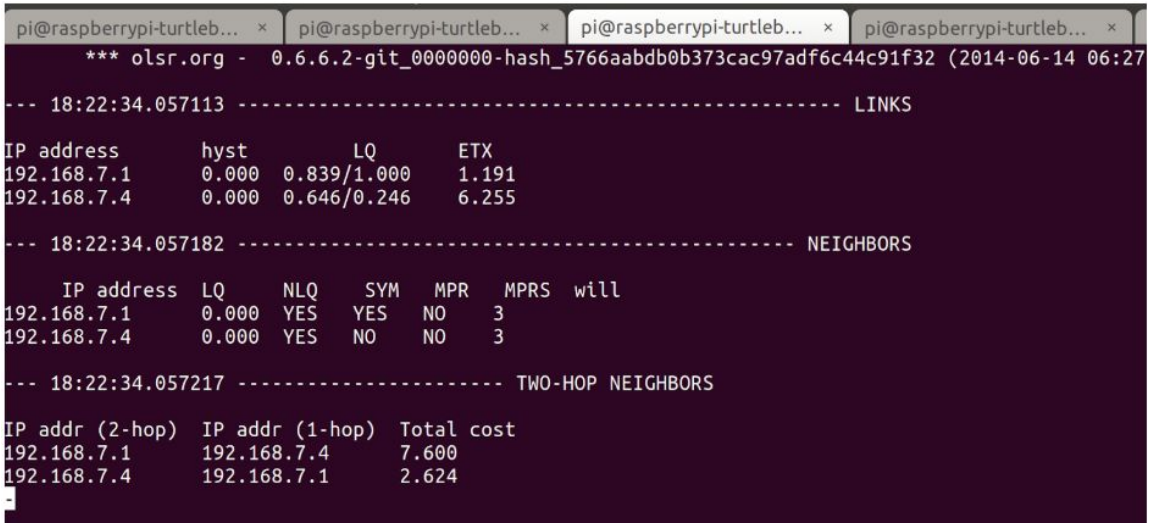

OLSR

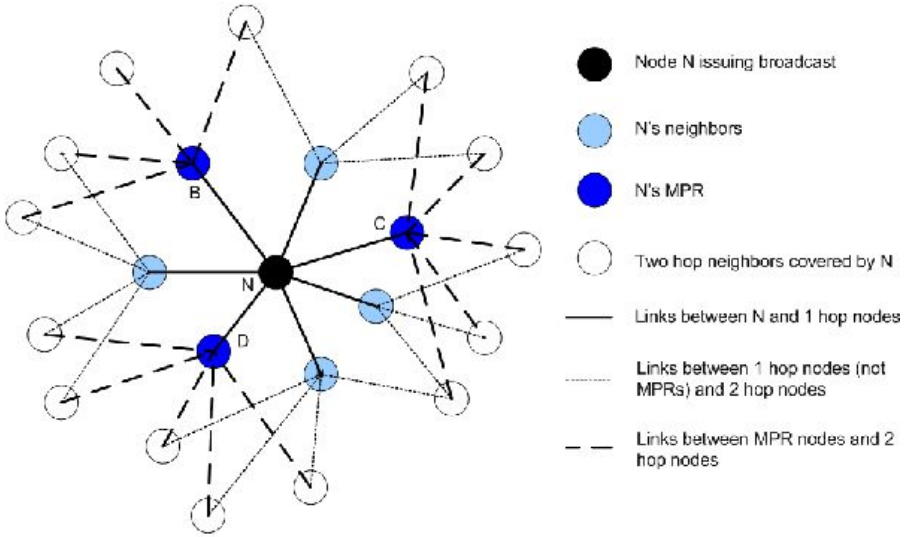
About OLSR

- Optimized Link State Routing Protocol
- Proactive Protocol
- Optimized



Some Terminology

- Multipoint Relay (MPR) -- the subset of your neighbors that forward your link states
- If X is your MPR, you are X’s MPR selector

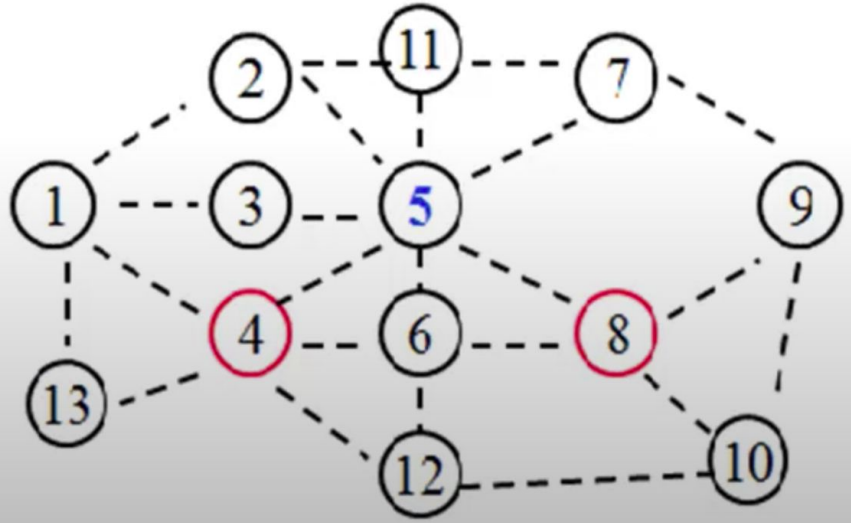


How it works

- Link State determined using ‘Hello’ messages
- MPR selected based on which one-hop neighbor provides best routes to two-hop neighbors
- Topology Control (TC) messages disseminate neighbor info
- Host and Network Association (HNA) messages spread route advertisements
- Only MPR’s forward Link State info
    - All nodes that receive the info use it to update their routing tables
- Routing Decisions use ETX = 1 / (LQ * NLQ) as a link metric

OLSR: Example

- Node 5 has selected 4, 8 as MPR
- Node 5 sends a LS to 2, 3, 4, 6, 7, 8, 11
- Nodes 2,3,6,7, 11 use the info but do not forward
- Nodes 4 uses the info and forwards it to 1, 6, 12, 13
- Node 8 uses the info and forwards it to 6, 9, 10



Taking a look at the code from OLSRd

- Load Inputs & Settings
- Initialization
- Start the **Scheduler**
- Close the program and clean up

Load Input and Settings

- CLI
- Command Line options, flags, other settings

Initialization

- Timers
- IP forwarding
- Route exporter
- Willingness to forward for others
- Interfaces
- IPC sockets
- Tables for displaying status of OLSR

Scheduler -- While (state == Running)

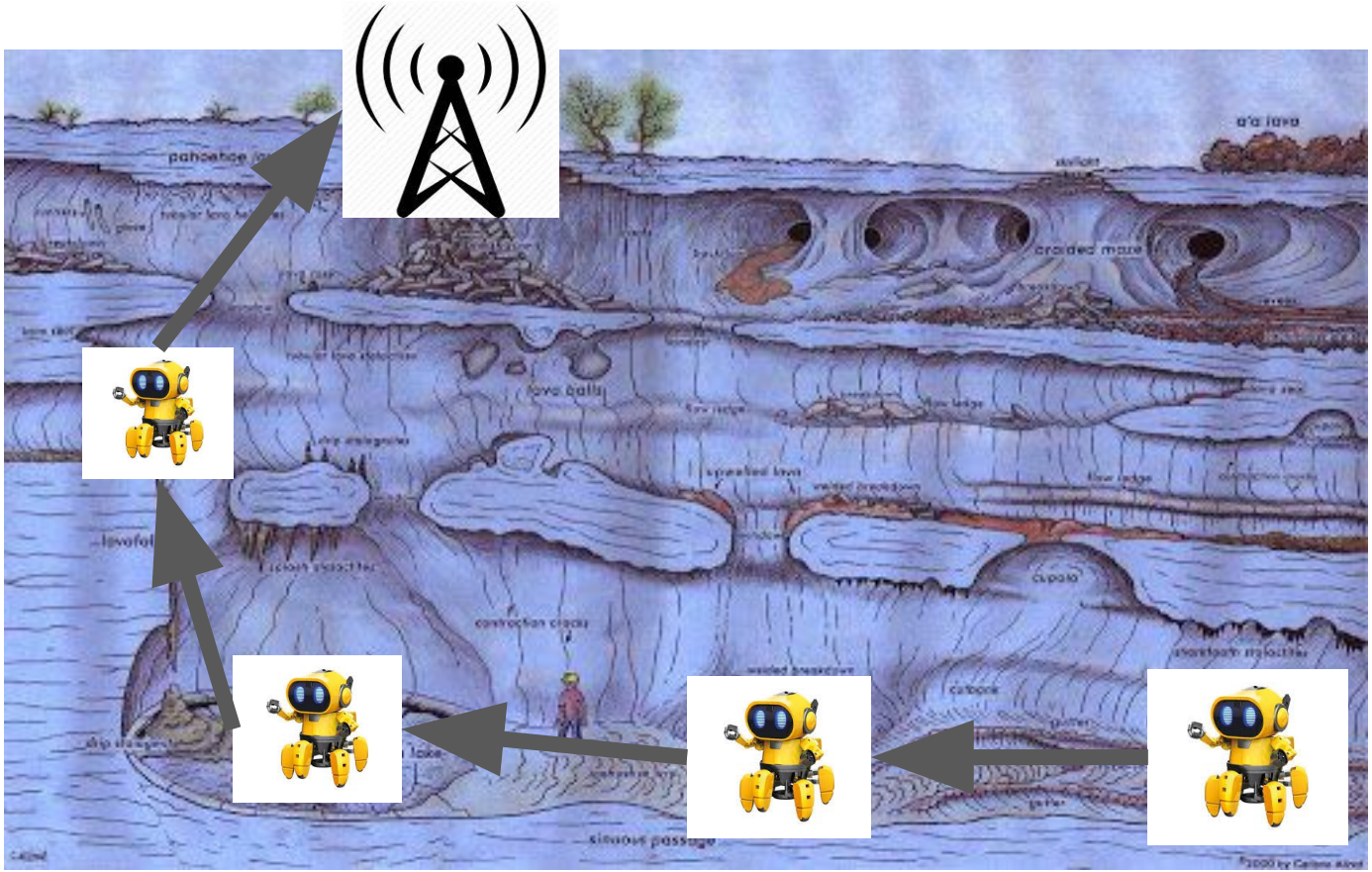
- Read in times as global timestamp
- Read incoming data
- Check for break condition (State != Running)
- Process Timers
- Check for break condition (State != Running)
- **Process Changes**
- Check for break condition (State != Running)
- If there were link changes, increment ANSN
- Check for break condition (State != Running)
- Handle incoming data immediately

Process Changes

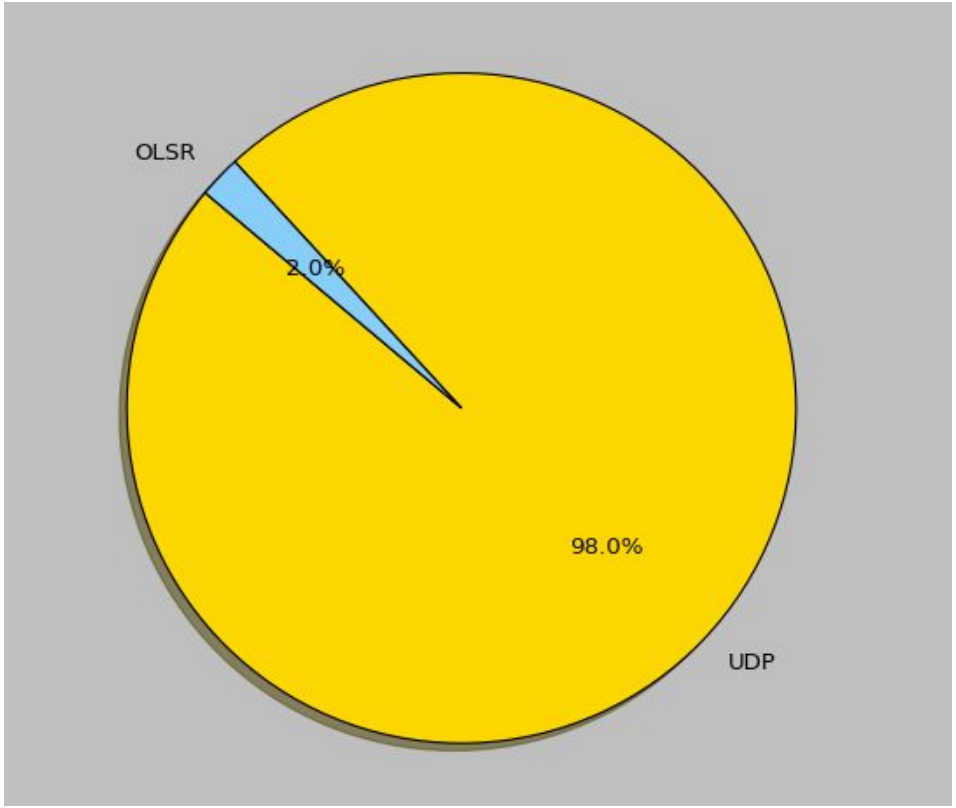
- Calculate the MPR’s
  - Willingness to forward messages
  - Network coverage
  - Limited overlap
- Calculate the Routing Table
  - Calculate edge weights to & from all neighbors
  - Calculate shortest path to neighbors using Dijkstra & Shortest Path First (SPF)
  - Send these routing changes to the Linux kernel

Simulation

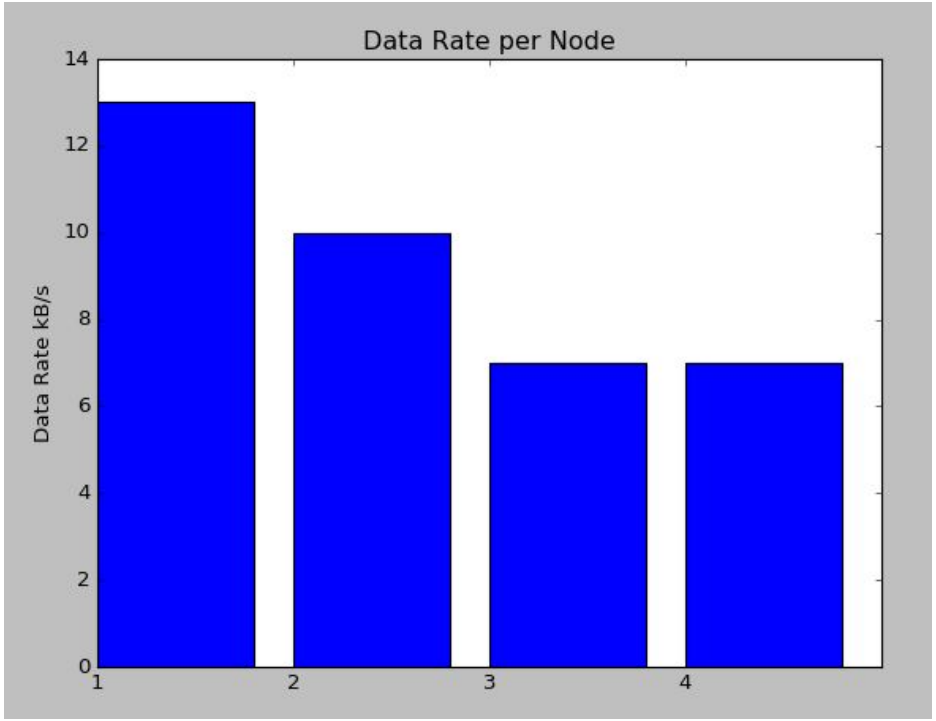
- UDP
- All OnOff Applications at 100kbps
- Link rates of 50Mbps

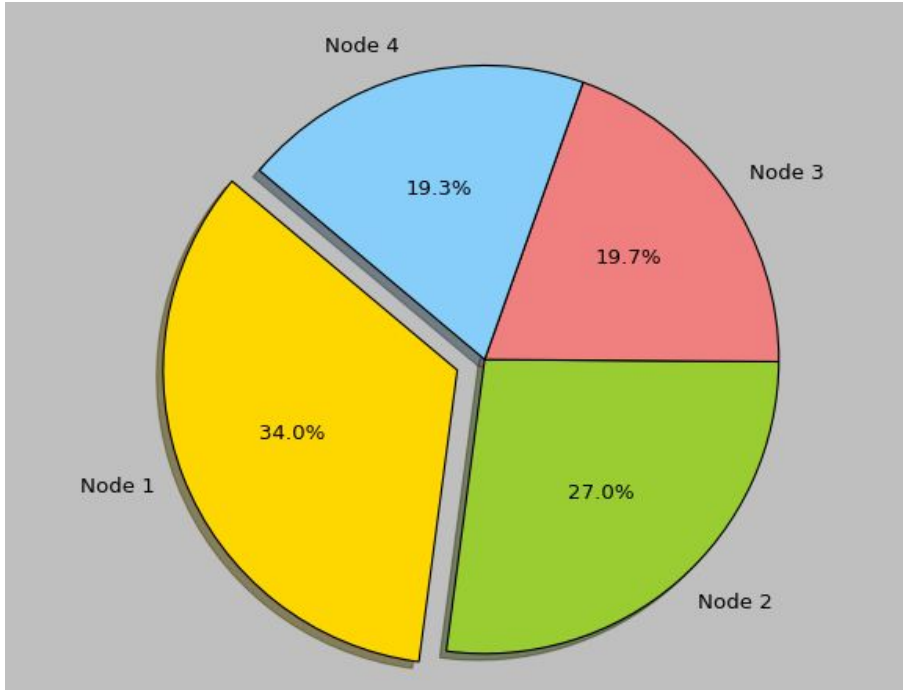


OLSR v UDP Packets

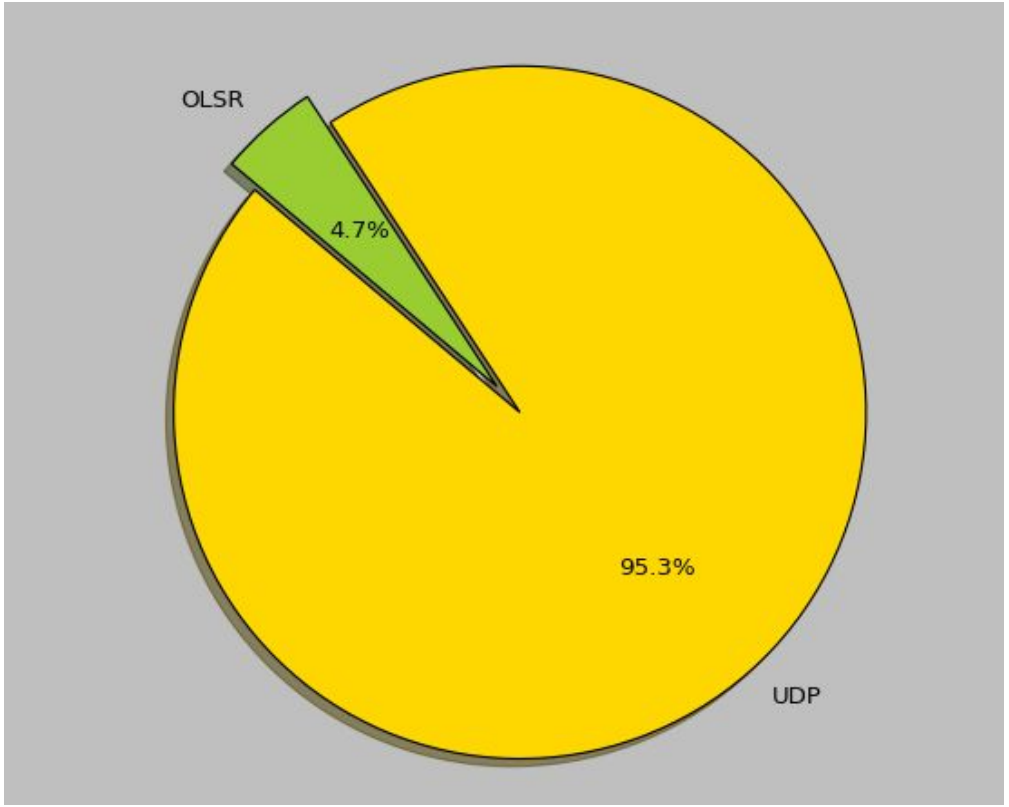


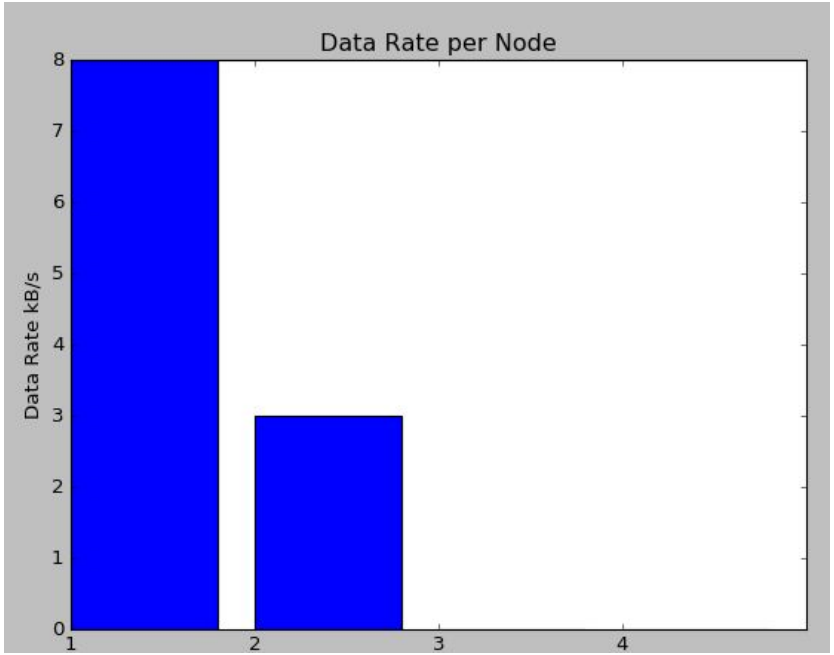
Data Rates


Data Rate per Node
Data Rate kB/s
0
2
4
6
8
10
12
14
1
2
3
4


Node 4
19.3%
Node 3
19.7%
34.0%
Node 1
27.0%
Node 2

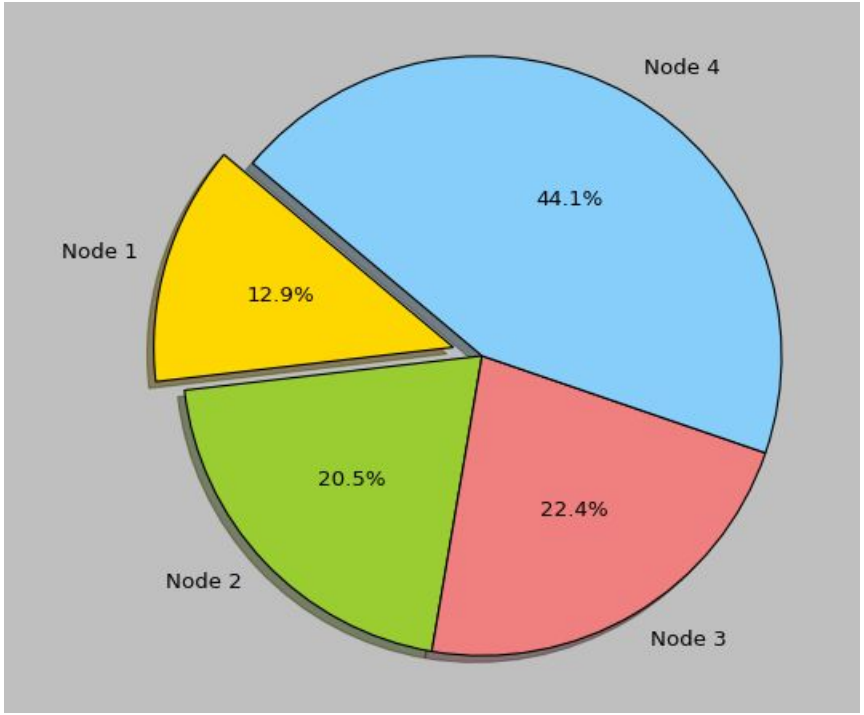
Made all data Rates equal and lower



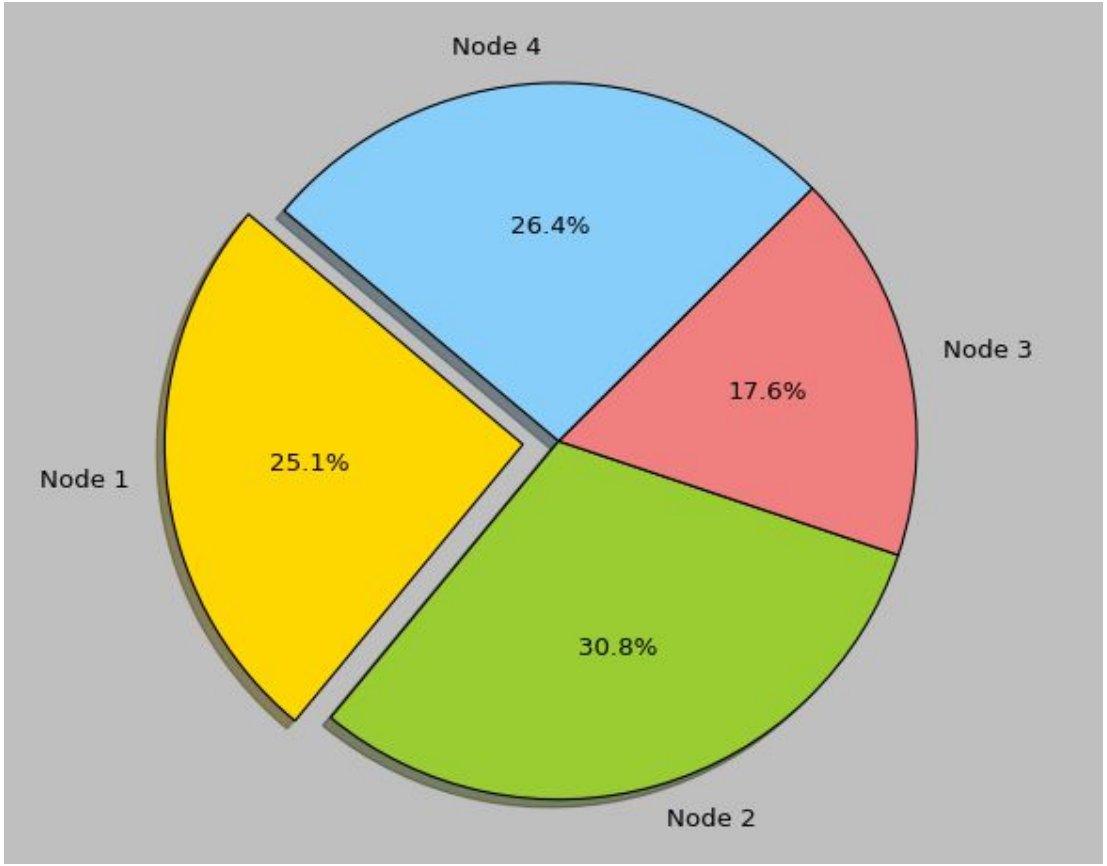


More BW from flows 3 & 4, high rate links

```
// 210 bytes at a rate of 100 Kb/s from n4 to n0
OnOffHelper onoff1 ("ns3::UdpSocketFactory",
                    InetSocketAddress (i01.GetAddress (0), port));
onoff1.SetConstantRate (DataRate ("600kb/s"));
ApplicationContainer onOffApp1 = onoff1.Install (c.Get (4));
onOffApp1.Start (Seconds (start_t));
onOffApp1.Stop (Seconds (sim_t+start_t));

// 210 bytes at a rate of 100 Kb/s from n3 to n0
OnOffHelper onoff2 ("ns3::UdpSocketFactory",
                    InetSocketAddress (i01.GetAddress (0), port));
onoff2.SetConstantRate (DataRate ("300kb/s"));
ApplicationContainer onOffApp2 = onoff2.Install (c.Get (3));
onOffApp2.Start (Seconds (start_t));
onOffApp2.Stop (Seconds (sim_t+start_t));

// 210 bytes at a rate of 100 Kb/s from n2 to n0
OnOffHelper onoff3 ("ns3::UdpSocketFactory",
                    InetSocketAddress (i01.GetAddress (0), port));
onoff3.SetConstantRate (DataRate ("200kb/s"));
ApplicationContainer onOffApp3 = onoff3.Install (c.Get (2));
onOffApp3.Start (Seconds (start_t));
onOffApp3.Stop (Seconds (sim_t+start_t));

// 210 bytes at a rate of 100 Kb/s from n1 to n0
OnOffHelper onoff4 ("ns3::UdpSocketFactory",
                    InetSocketAddress (i01.GetAddress (0), port));
onoff4.SetConstantRate (DataRate ("100kb/s"));
ApplicationContainer onOffApp4 = onoff4.Install (c.Get (1));
onOffApp4.Start (Seconds (start_t));
onOffApp4.Stop (Seconds (sim_t+start_t));
```



Same demands per flow, slower links



Same Demands, very slow links ~50kbs