The Greatest Network Simulation Final Presentation Of All Time

Hongjian L
Yamei O
Samuel R
Jan VB
Junlin Z

12/02/2014

Outline

- ➢ Background
- ➢ Our Team
- ➢ Architecture
- ➢ Routing Algorithms
- ➢ Transport Layer Algorithms
- ➢ Results of Test Cases
- ➢ Extra Experimentation

Background

- Abstract network simulator
- Input - Network description
- Simulate - For a user-specified duration
- Record - User-specified data at events
- Output - Graphs of data over time

Our Team

- ➢ Hongjian Lan - Hosts, Routers
- ➢ Yamei Ou - Routers, Routing Alg’s
- ➢ Samuel Richerd - Inputs, Links
- ➢ Jan Van Bruggen - Manager, Outputs
- ➢ Junlin Zhang - Flows, Transport Layer Alg’s

Architecture

- ➢ Python and SimPy
- ➢ Controller
- ➢ Input File Structure
- ➢ Events
- ➢ Actors

Architecture - Controller

- Reads input file
- Creates simulation environment
- Creates actors
- Runs simulation
- Graphs outputs

Architecture - Input File

- Custom file structure
  - Human-readable (easy to edit)
  - Comments
  - Reusable object parameters (for simpler files)

```
LINK
    ID        L1
    RATE      10
    DELAY     10
    BUFFER    64
    CONNECTS R1  R2
    ID        L2
    CONNECTS R1  R3
```

Architecture - Events

- ➢ Dynamically generated by actors
- ➢ As few events as possible at any time
- ➢ Call specific actor reactions upon completion

Architecture - Host

- Receives packets from flows
  - Hands them to its link
- Receives packets from link
  - Hands them to their flow.

Architecture - Flow

- Makes data packets
  - Hands them to source host
- Receives data packets from destination host
- Makes ack packets
  - Hands them to destination host
- Receives ack packets from source host
  - Hands them to transport layer algorithms

Architecture - Link

- Paired one-way links model full-duplex links
  - Every link in input file becomes two (A & B)
- Receives packets from host or router
  - Adds packet to buffer
- Transmits packets
  - Schedules “Packet Receipt”, “Link Available” events
- Reacts to “Link Available” events
  - Transmits new packet

Architecture - Buffer

- Buffer receives packets from link
  - Drops packet if buffer is full
  - Defined by number of bytes

Architecture - Router

- Receives packets from links
  - Immediately hands them to another link
- Updates routing table periodically

Algorithms-Routing

- ➢ Routers implement Bellman-Ford
- ➢ User-specifiable link cost metrics
  - ○ Static routing: number of hops
  - ○ Dynamic routing: packet delay

Algorithms-Dynamic Routing

- ➢ Generate router packet
  - ○ Send router packet to their neighbors
  - ○ Router packet creates start timestamp
- ➢ Generate acknowledgement of router packet
  - ○ Send acknowledgement back to original router after receiving router packet
  - ○ Acknowledgement packet contains a copy of router table and keeps start timestamp

Algorithms-Dynamic Routing

- ➢ Receive acknowledgement of router packet
  - ○ Update routing table based on the received router packets' timestamps
- ➢ Need to find proper time intervals to generate router packet for different cases.

Algorithms-TLA

- ➢ Transport Layer Algorithm
- ➢ Tahoe-Base
  - ○ Tahoe
  - ○ Tahoe with fast retransmit
  - ○ Reno
- ➢ Vegas-Base
  - ○ Vegas
  - ○ FAST

Algorithms-TLA

- Tahoe-Base
    - param: enable_fast_retransmit
    - param: enable_fast_recovery
  - react_to_flow_start
  - react_to_ack
    - w=w+1 (w<ssthresh)
    - w=w+1/w (w>ssthresh)
    - duplicate ack: fast retransmit / fast recovery
    - calculate rtt_avg, rtt_div
  - react_to_time_out (rtt_avg + 4 x rtt_div)
    - ssthresh=w/2
    - w=1
    - start slow start

Algorithms-TLA

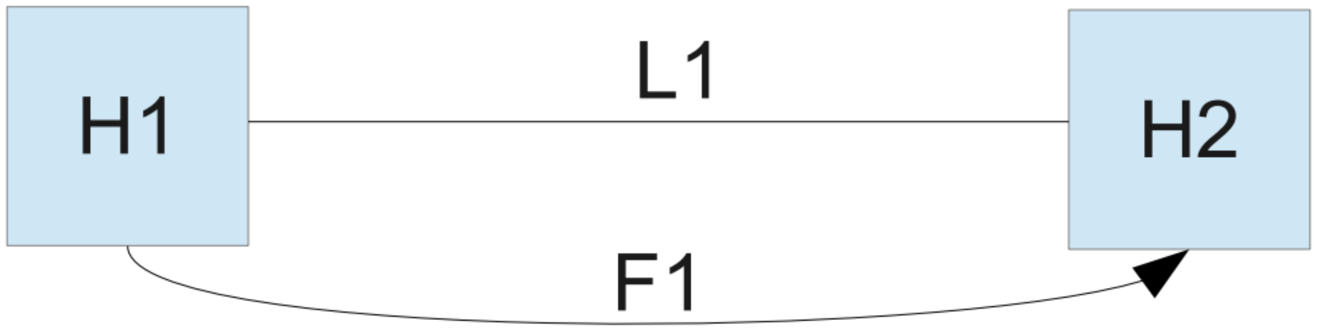
- Vegas-Base
    - param: enable_fast
  - react_to_flow_start
  - react_to_ack
    - w=w+1 $(w/rtt_{min}-w/rtt<\gamma/rtt_{min})$
    - calculate rtt_avg, rtt_div
  - react_to_time_out
    - w=w/2
  - react_to_vegas_update (every rtt)
    - Vegas:
      - w++ $(w/rtt_{min}-w/rtt<\alpha/rtt_{min})$
      - w−− $(w/rtt_{min}-w/rtt>\beta/rtt_{min})$

Algorithms-TLA

- Vegas-Base
    - param: enable_fast
- react_to_flow_start
- react_to_ack
    - $w=w+1$ $(w/rtt_{min}-w/rtt<\gamma/rtt_{min})$
    - calculate rtt_avg, rtt_div
- react_to_time_out
    - $w=w/2$
- react_to_vegas_update (every rtt)
    - FAST:
        - $w=rtt_{min}/rtt \cdot w+\alpha$

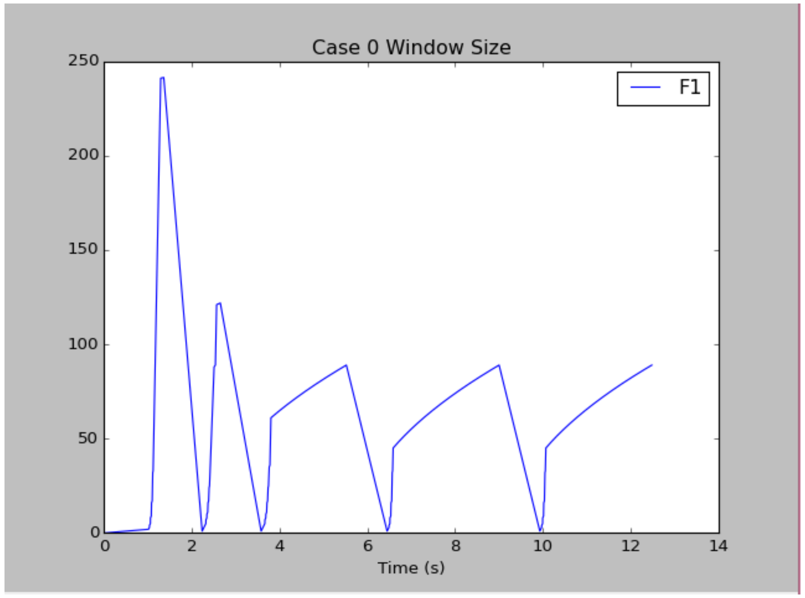
Results of Test Cases

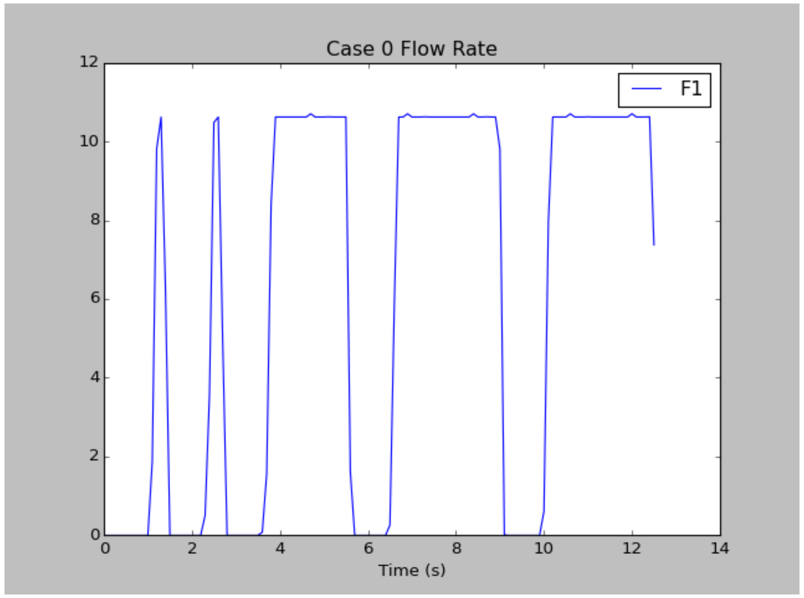
Test Case 0



Results of Test Cases

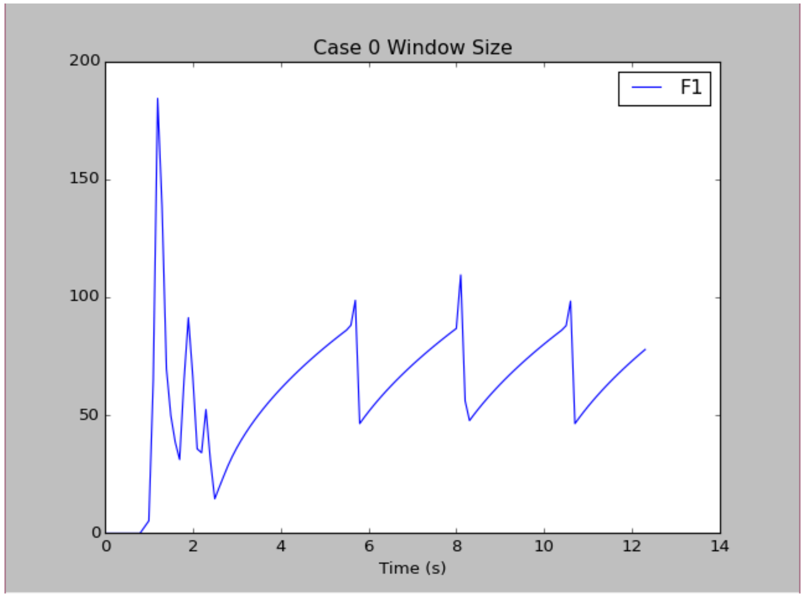
Test Case 0: Tahoe

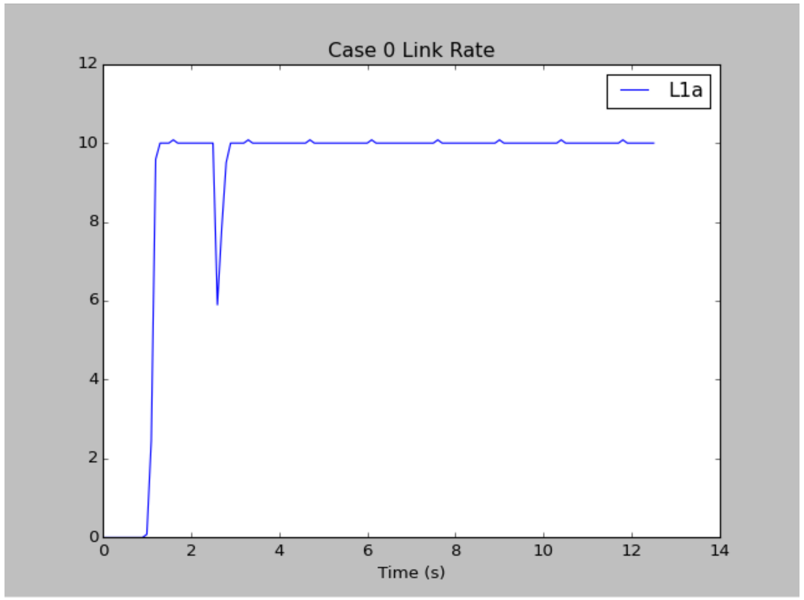




Results of Test Cases

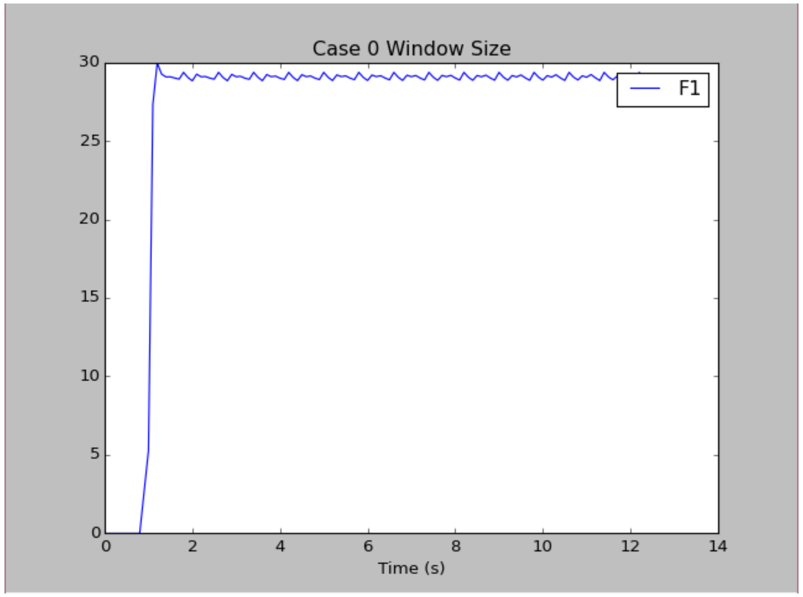
Test Case 0: Reno

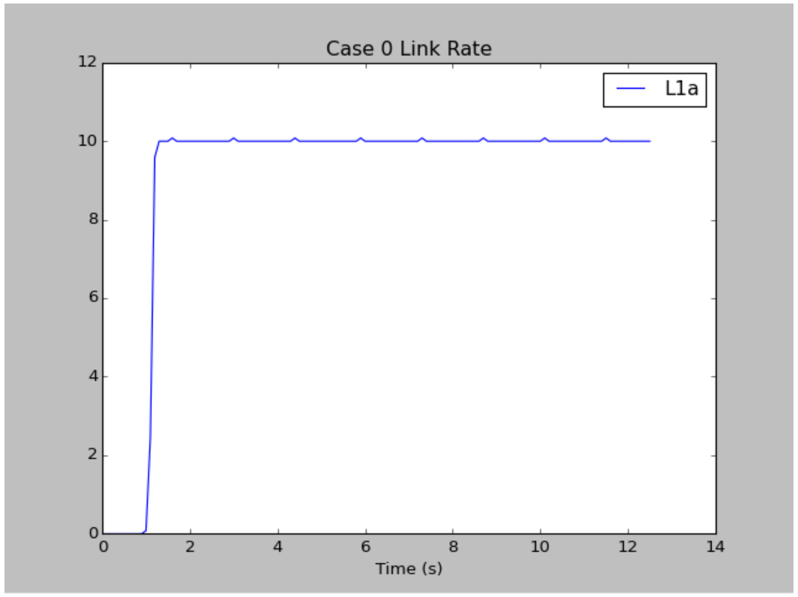




Results of Test Cases

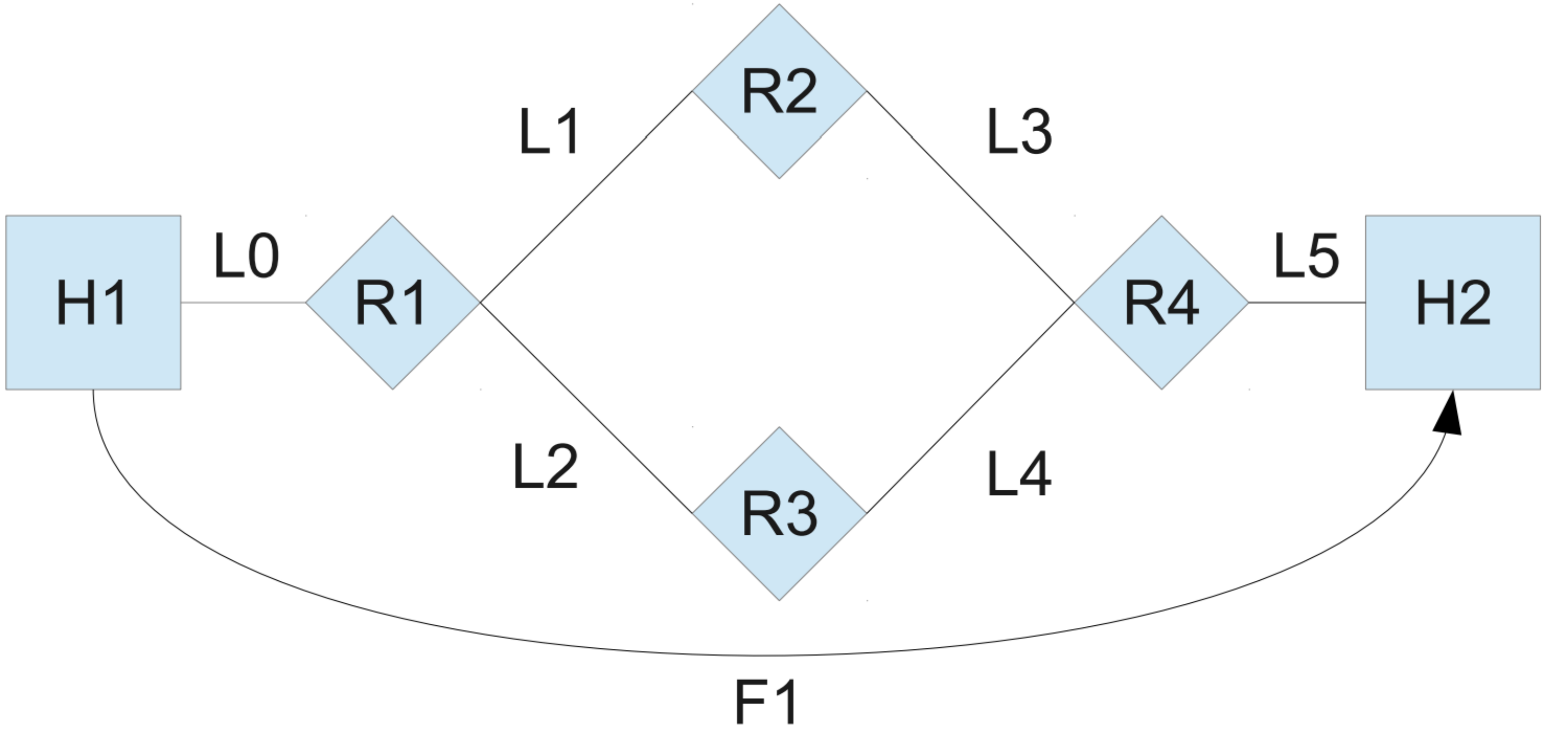
Test Case 0: FAST





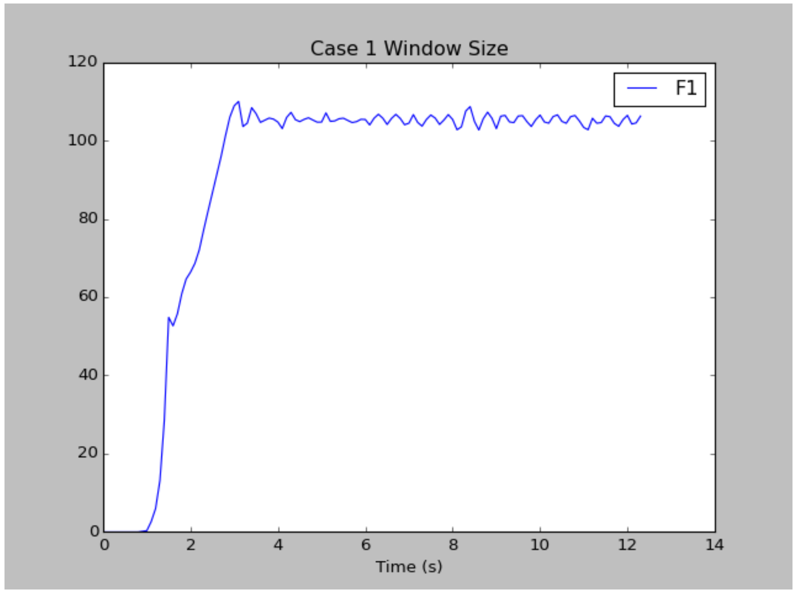
Results of Test Cases

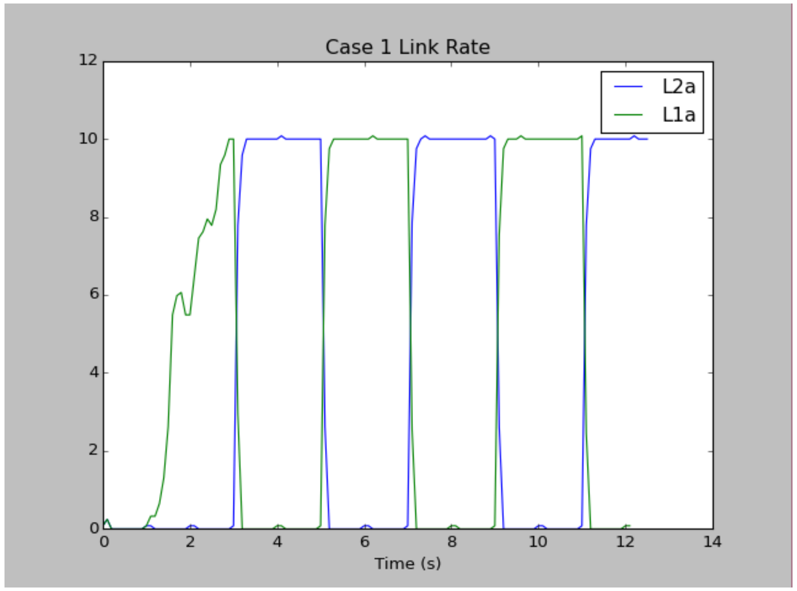
Test Case 1



Results of Test Cases

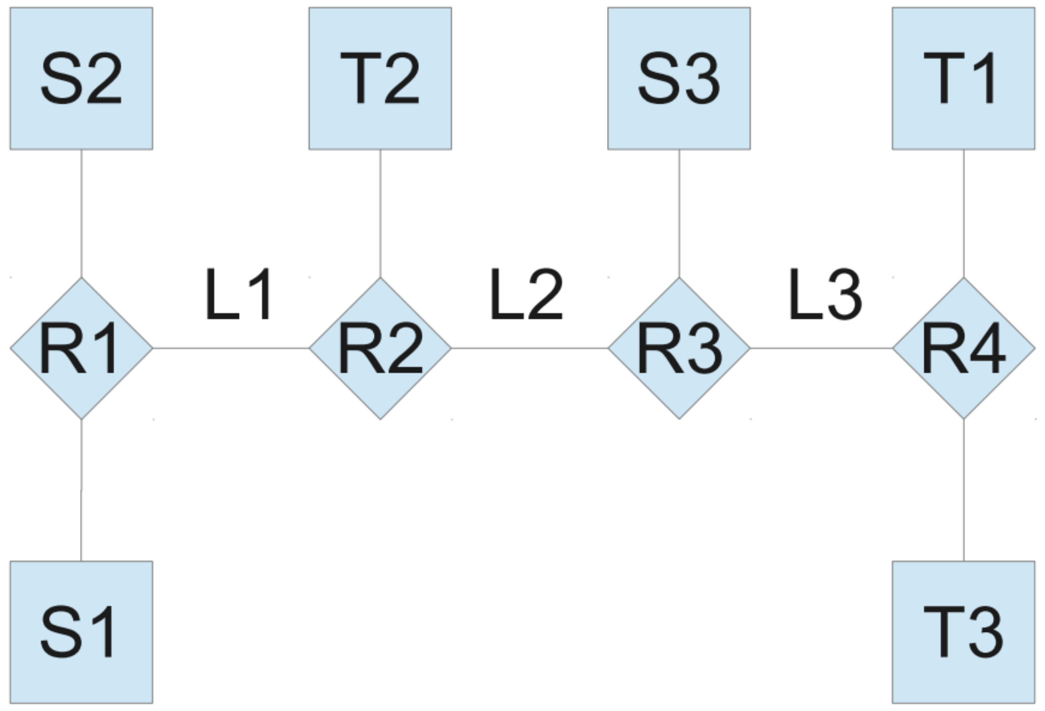
Test Case 1: FAST





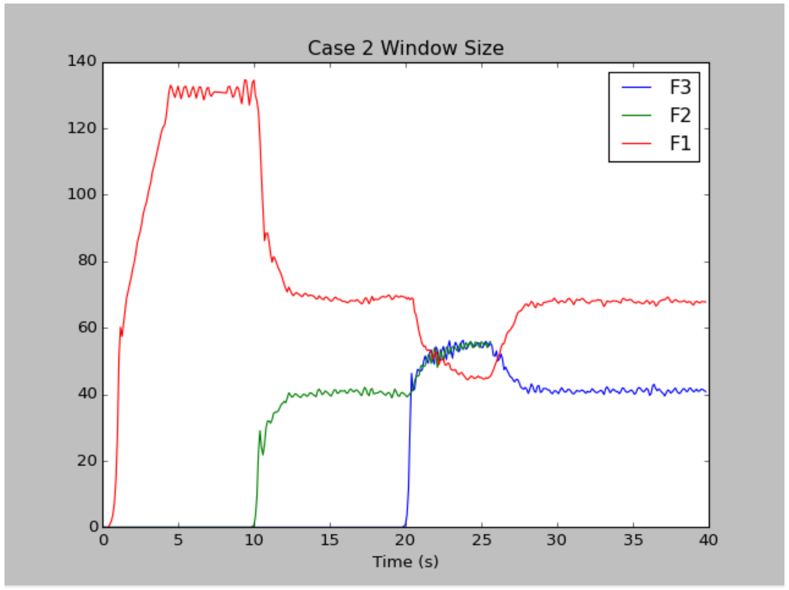
Results of Test Cases

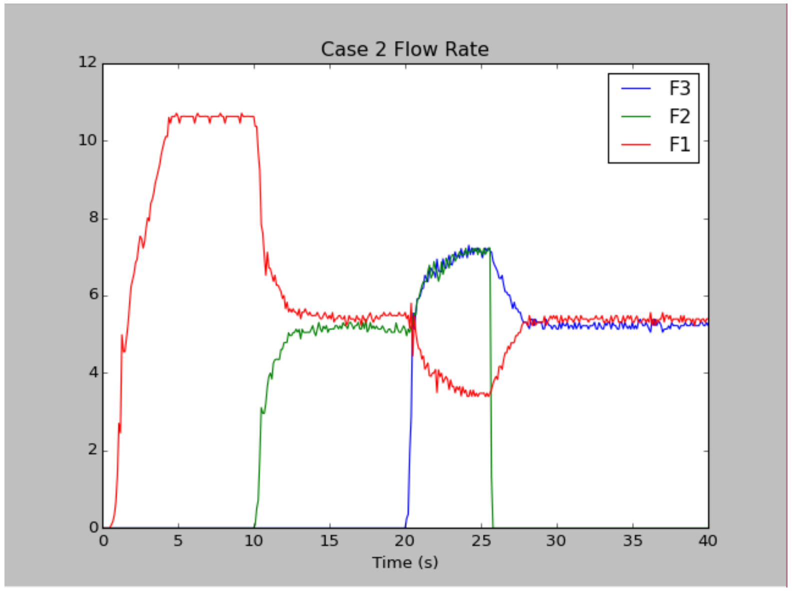
Test Case 2



Results of Test Cases

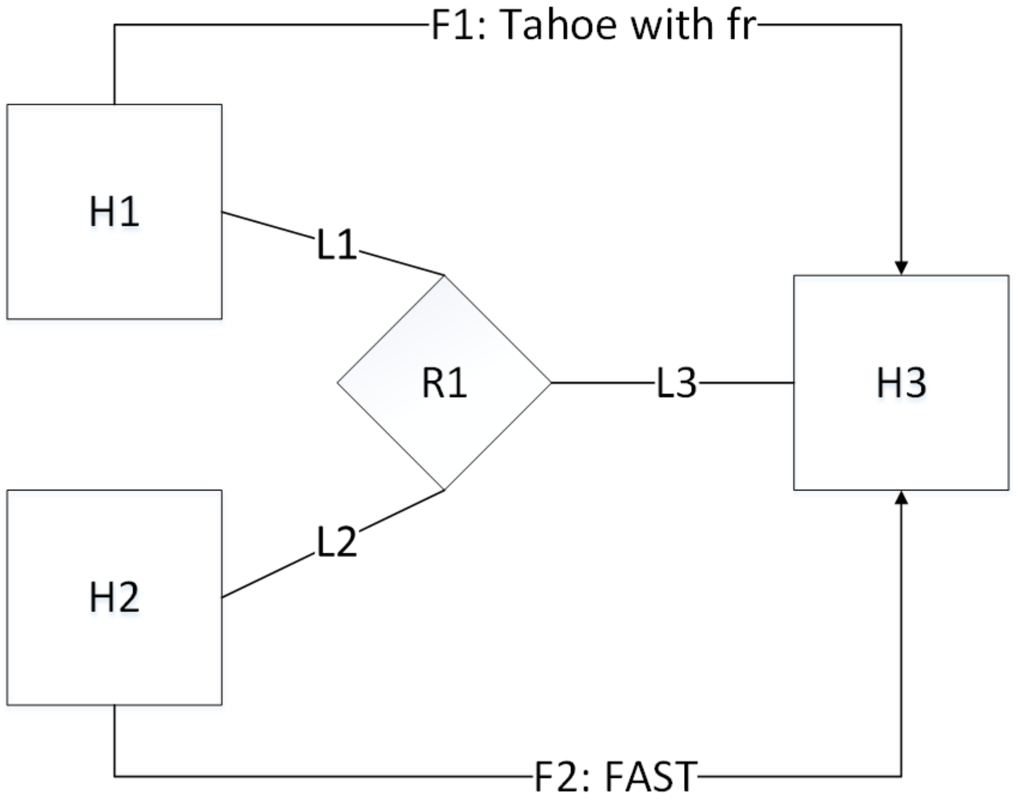
Test Case 2: FAST





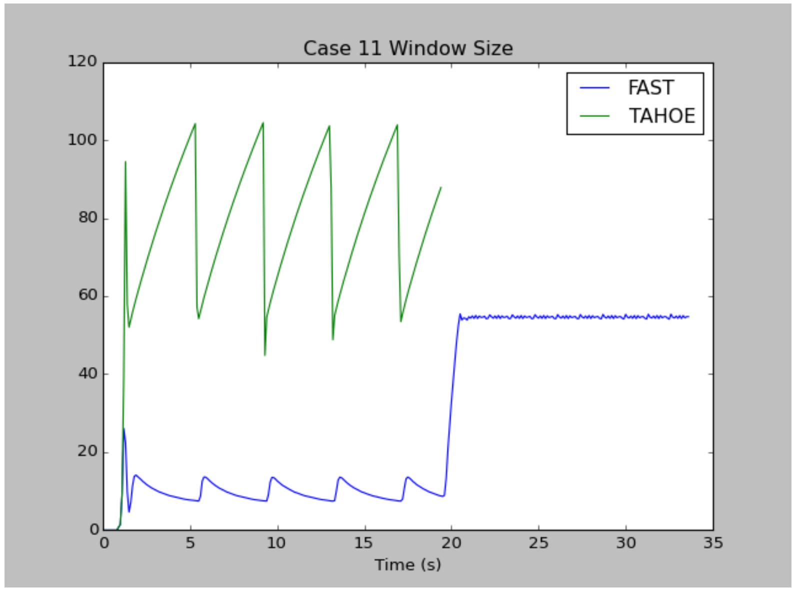
Extra Experimentation

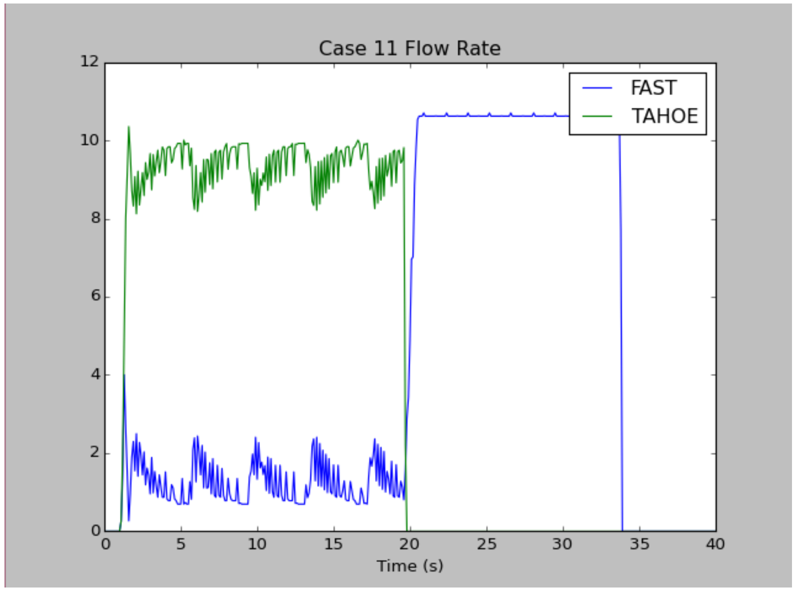
➢ Different protocols running together



Extra Experimentation

F1: Tahoe (with fr)   vs.   F2: FAST





Questions?