**CS 143 Simulation Final Report**

**Network Simulation**

Ruijia Sun, Leiya Ma, Chenshuo Yue

12/09/2015

**0. Division of Labor**

Ruijia: Host, Link, Packet, Transport Layer Algorithm, Flow Experimentation, Test Cases (C++)

Leiya: Input File Editor, Parser, Initialization, Router Algorithm, Presentation, Report (C++)

Chenshuo: Output Plotter (Matlab)

**1. Project Roadmap**

**Milestone 1 (Week 5)**

**Progress**

* Chose C++ as the Langrage.
* Decided actors (Flow, Host, Link, Router, Packet) and implement class definitions.
* Using Half-Duplex Mode.
* Decided Reno and FAST TCP Algorithm.
* Decided the units and constants.

**Milestone 2 (Week 8)**

**Progress**

* Implemented static routing algorithm.
* Used one link with two buffers on different sides to simulate half-duplex mode.
* Finalized our input structure.
* Ran the test case 0 and 1 and the outputs seemed correct.
* Implemented Reno.

**Milestone 3 (Week 10)**

**Progress**

* Implemented the Bellman-Ford algorithm.
* Fixed the bug of multiple packet losses during one RTT.
* Refined the way the routers update their routing tables (sending an update request to the router’s neighbors and waiting for replies) so the result of test case 1 was correct.
* Implemented FAST TCP.
* Drew graphs of outputs on Matlab.
* Changed to full-duplex work mode (The queueing delay was correct) so the FAST works correct as it uses queueing delay as a congestion measure.

**Milestone 4 (Week 11)**

**Progress**

* Cleaned up our code.
* Wrote more documentation.
* Finished this final report.

**2. Running the Simulation**

The total project can be divided into two parts according to different execute environment.

* The first part is running under C++ using Xcode Compiler. This part will load in txt file and simulate the whole network and finally generate data files of the simulation results.
  + First open code/Network/Network.xcodeproj
  + Then edit schema (Product/Schema/edit Schema): add Argument “test1.txt” or “test2.txt” to link and parse in testcase1 input file and testcase2 inputfile; change Work Directory to “outcome” file.
  + You can change the TCP algorithm (Reno or FAST) by changing the bool MODE in line 50 main.cpp. Set MODE = true, we have Reno; Set MODE = false, we have FAST.
  + Run project and you can see generated txt file (e.g. LinkBuffer4.txt) in our “outcome” file.
* The second part is running under Matlab plotting our generated file of part1. There are two .m files named “plot\_case1.m” and “plot\_case2.m” which you can use to plot testcase1 and testcase 2 respectively.

**3. Input Parsing**

We chose to create a custom input file structure (rather than use a common format such as JSON) because we wanted to make the input file easily editable. This ease of use of the input greatly increases the usefulness of our network simulator since it allows a user to rather intuitively interact with the input file parser.

**Format:**

The input file itself is must be a text file, and will be parsed on a line by line basis. Each line contains an Actor Entity to parse. Different kind of Actor Entity are divided by empty line.

**Entity Keywords:**

* flow - Flows require ID, SRCH, DSTH, SRCR, DSTH, DATA AMOUNT, START TIME attributes.
* node – Node require ID, nodeType (HOST or ROUTER).
* link - Links require ID, RATE, DELAY, LEFT\_CONNECTED NODE, RIGHT\_CONNECTED NODE attributes.
* host – Hosts require ID, FLOW ID, LINK ID, IS\_LEFT attributes.
* port – Ports require ID, next Node ID, outway LINK ID and IS\_LEFT attributes.
* router - Router requires ID and its ports vector (indicate port ID in vector).

**Example:**

flow 0 0 1 0 3 20000 500

node 0 HOST

node 1 HOST

node 0 ROUTER

node 1 ROUTER

node 2 ROUTER

node 3 ROUTER

link 0 12500 10 524288 0 2

link 1 10000 10 524288 2 3

link 2 10000 10 524288 2 4

link 3 10000 10 524288 5 3

link 4 10000 10 524288 5 4

link 5 12500 10 524288 1 5

host 0 0 0 0

host 1 0 5 0

port 0 0 0 1

port 1 3 1 0

port 2 4 2 0

port 3 2 1 1

port 4 5 3 1

port 5 2 2 1

port 6 5 4 1

port 7 1 5 1

port 8 3 3 0

port 9 4 4 0

router 0 0 1 2

router 1 3 4

router 2 5 6

router 3 7 8 9

**4. Host**

**Class (Main Members)**:

* Address (Unique IP Address to identify hosts)
* Flows (List of flows on this host)
* Link (The link connected to this host)

**Host is mainly in charge of sending and receiving packets and execute TCP algorithm.**

1. send( )

* Generate a new packet.
* Add the packet received to its link’s buffer.

2. cControl ( )

* Execute TCP algorithm: New Reno and FAST
* Take a “state” string as an input, and use it to differentiate situations:
  + “Received in Normal” : adjust the window size accordingly
  + “Received in FR”: window size ++
  + “Exit FR without TO”: set the window size to the ssthresh
  + “3Dup”: execut FAST retransmit.
    - ssthresh = max(wndSize/2.0, 1.0);
    - wndSize = ssthresh + 3;
  + “TimeOut”: set the status back to slow start
    - ssthresh = max(wndSize/2.0, 1.0);
    - wndSize = 1;
    - status = "ss";

**5. Link**

We simulate a two-way link by two one-way links, one from the source of the two-way link to its destination, the other from the destination of the two-way link to its sources.

**Class**:

* Source & Destination (Link’s source & destination)
* Delay (The propagation delay of the link)
* Rate (Link’s link rate)
* Buffer (Link’s buffer)

**Link is in charge of sending packets from its source to its destination**.

1. send( )

* Divide the packet’s size by link rate to get the transmission delay. Add transmission delay and propagation delay together to get the total delay for the packet.
* Call HostReceive( ) and RouterReceive(). Receive the packet after the total delay.
* Call LinkAvailable( ). The link will be available again after the transmission delay.

**6. Flow**

**Class:**

* Source Host & Destination Host
* Source Router & Destination Router
* Total packet amount
* Remain packet amount
* Starting time

**Link is not in charge of TCP algorithm.**

**7. Router**

**Class**:

* Address(Unique IP address to indentify routers)
* Ports (list of ports to this router) indicate which link and node are connected to this port

**Router Initialization**:

We mainly need to initialize router table. When initializing router table we create spaces for every host in testcases, set their distance to be 1 if the host is connected to the router, otherwise to be infinity if there is no direct connection.

We will generate first bunch of router packet immediately after initializaion.

We assume that there is enough time to update router table before first flow starts.

**Router is mainly in charge of two tasks**:

1. React to event RouterReceive
2. React to event RouterUpdate
3. React to event RouterTimeOut

Explain in more details:

* React to event RouterReceive:

This is a reaction after receiving a packet from links in simulation.

There are several steps following:

1. Tell what type of the packet is
   1. DataPacket==>forward()
   2. RouterPacket==>send() and updateRT()
   3. ACK of RouterPacket==>send()
2. Call different funtions according to type of packet
   1. forward(): Get the destination of packet, then look up current router table, find next hop and send the packet to next link.
   2. send(): This is a function to react after a neighbor router receiving a routerpacket, it will generate an ACK packet which also contains current routing information.
   3. computeCost(): This is a function to compute the costInfo of the routes currently. There are two steps: First loop through neighbor routers to compute and update the costInfo. Then look through routing table to compare route which does not include current neighbors. If new route costs less, update routing tables’ nexthop.
   4. updateRT(): Compare costInfo from Costpkt received from neighbor routers and the current router. Update router table if there is a shorter path. The rule of update is Bellmen-Ford algorithm which we will explain later.

* React to RouterTimeOut:

Retransmit and using send()

**Routing algorithm: Bellmen-Ford algorithm**

Static routing: cost based on the maximum number of packets on link

Dynamic routing: number of packets in queueing buffer

**9. Events**

HostTimeOut

HostReceive

LinkAvailable

RouterTimeOut

* Retransmit the costInfo packet

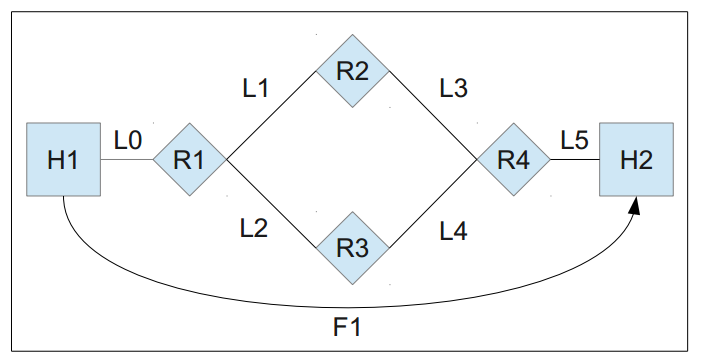
这里解释一下Receive 和 Update 的区别，就是presentation的时候学长问的那个问题

RouterReceive()

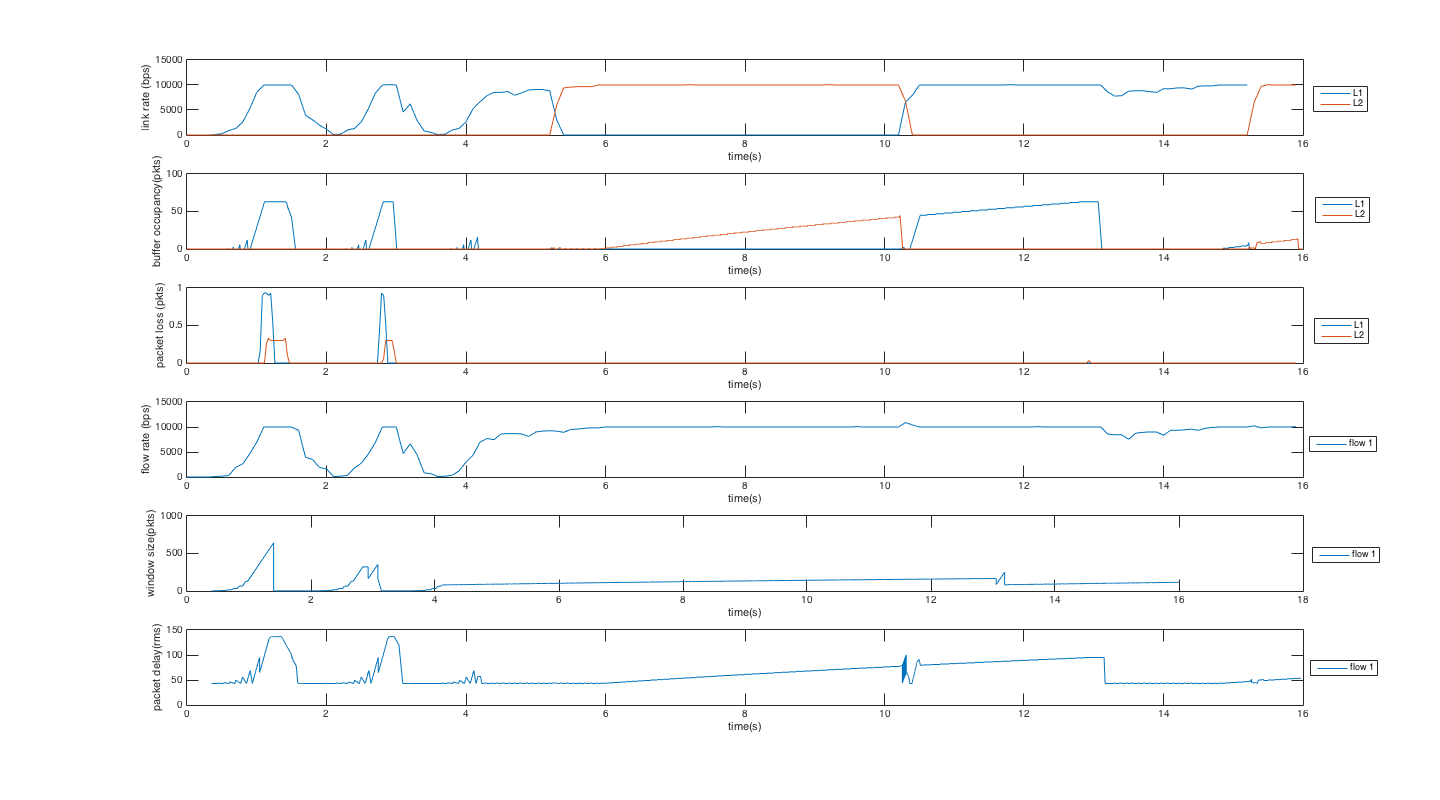
RouterUpdate()

**11. Test Cases and Analysis**

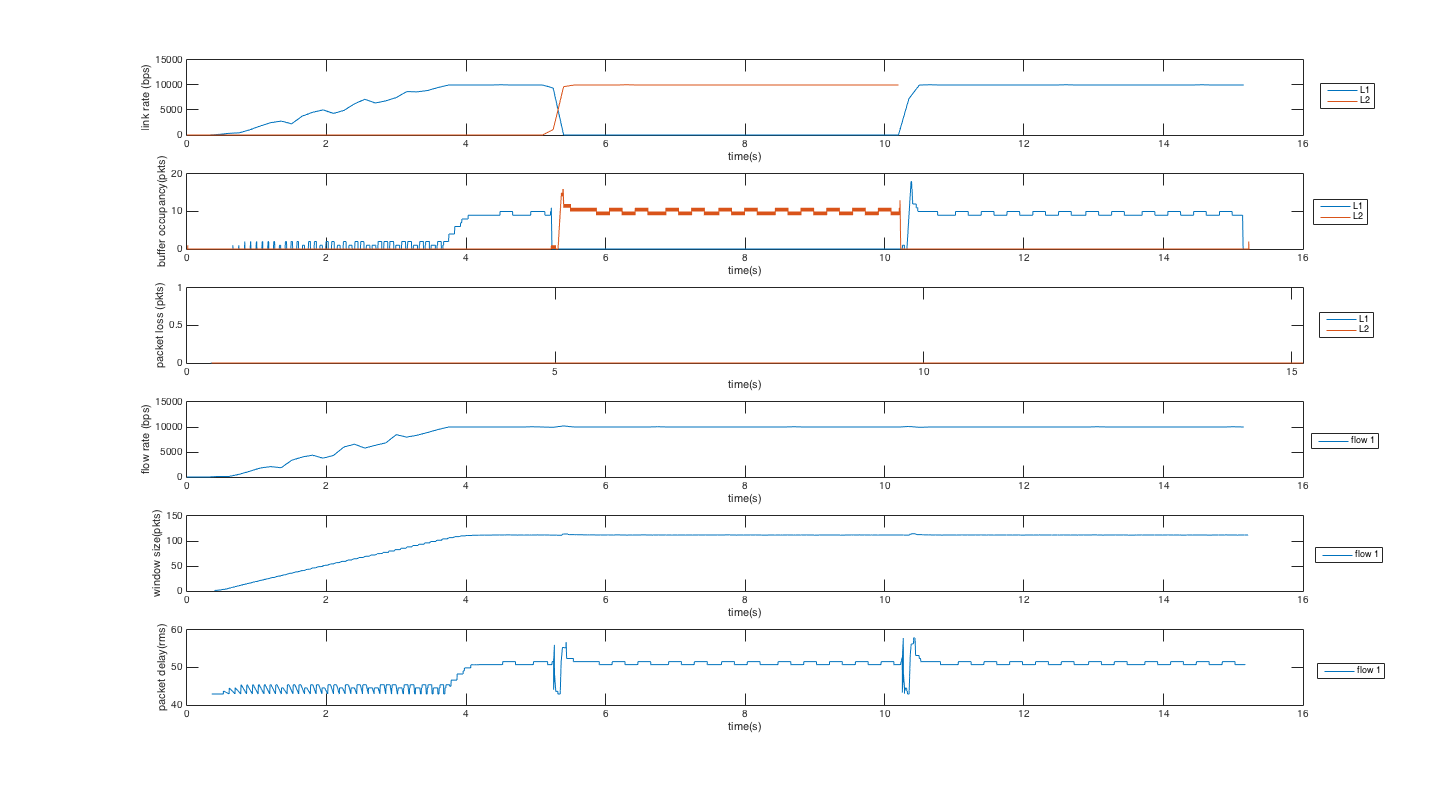
Test Case 1



RENO:



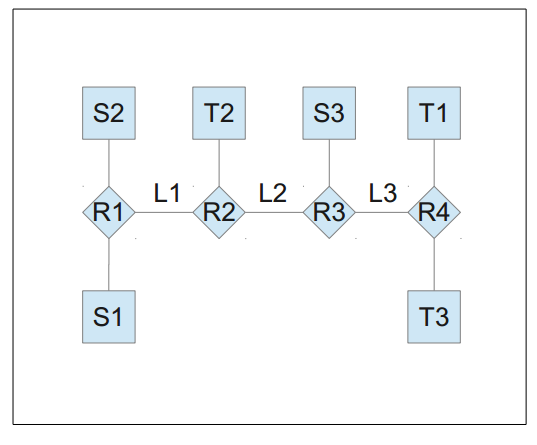
FAST:



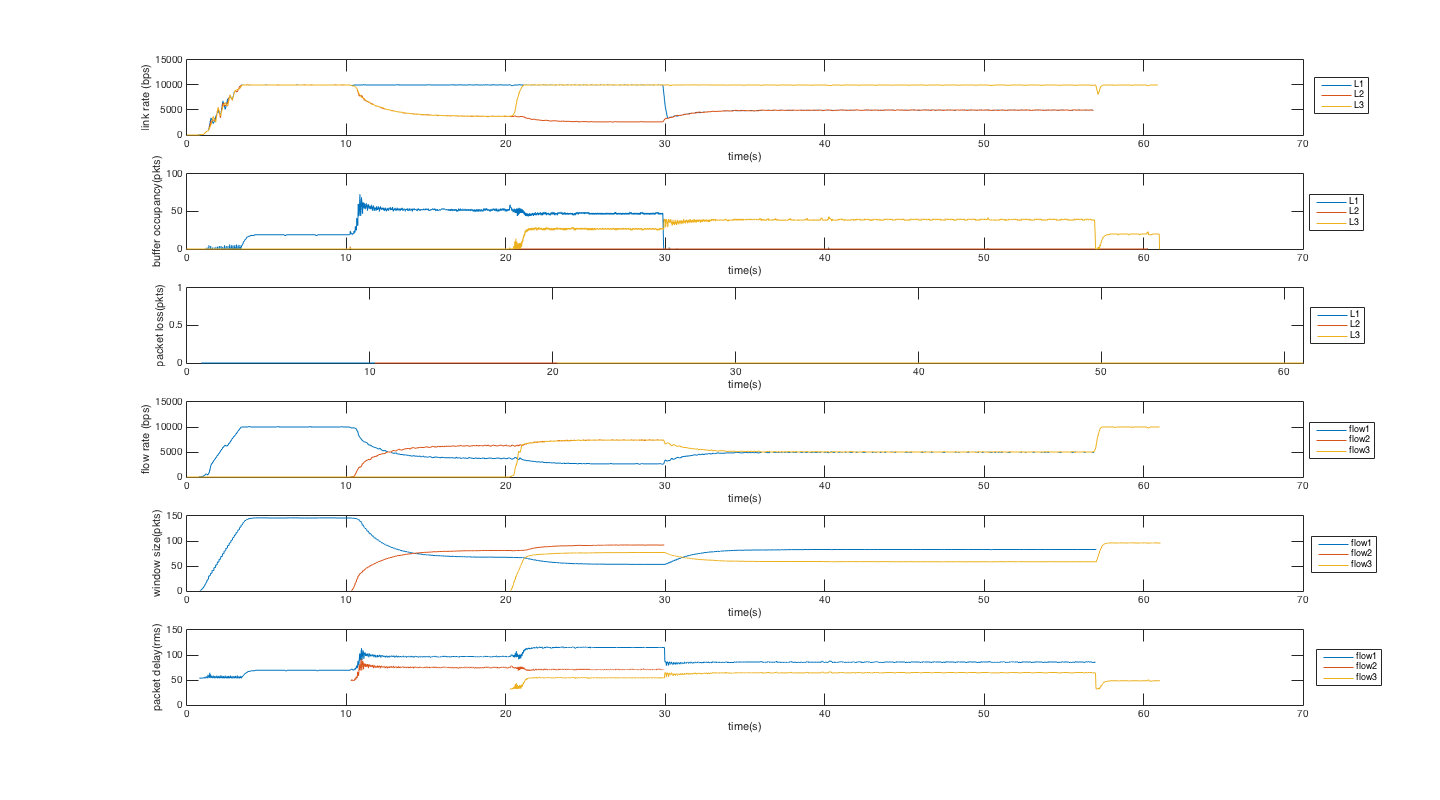
we analysis some issues related to dynamic routing (based on FAST TCP, ). When dynamic routing is used, the flow change the route between H1->R1->R2->R4->H2 and H1->R1->R3->R4->H2, so we can see the link rate of L1 and L2 are exchanged periodically. It should not make much difference to the flow rate. So it will still take around 16 seconds to transmit the data.

We can see that there is some oscillation of buffer size. It might be caused by that before it runs into a more stable state, the routing has changed.

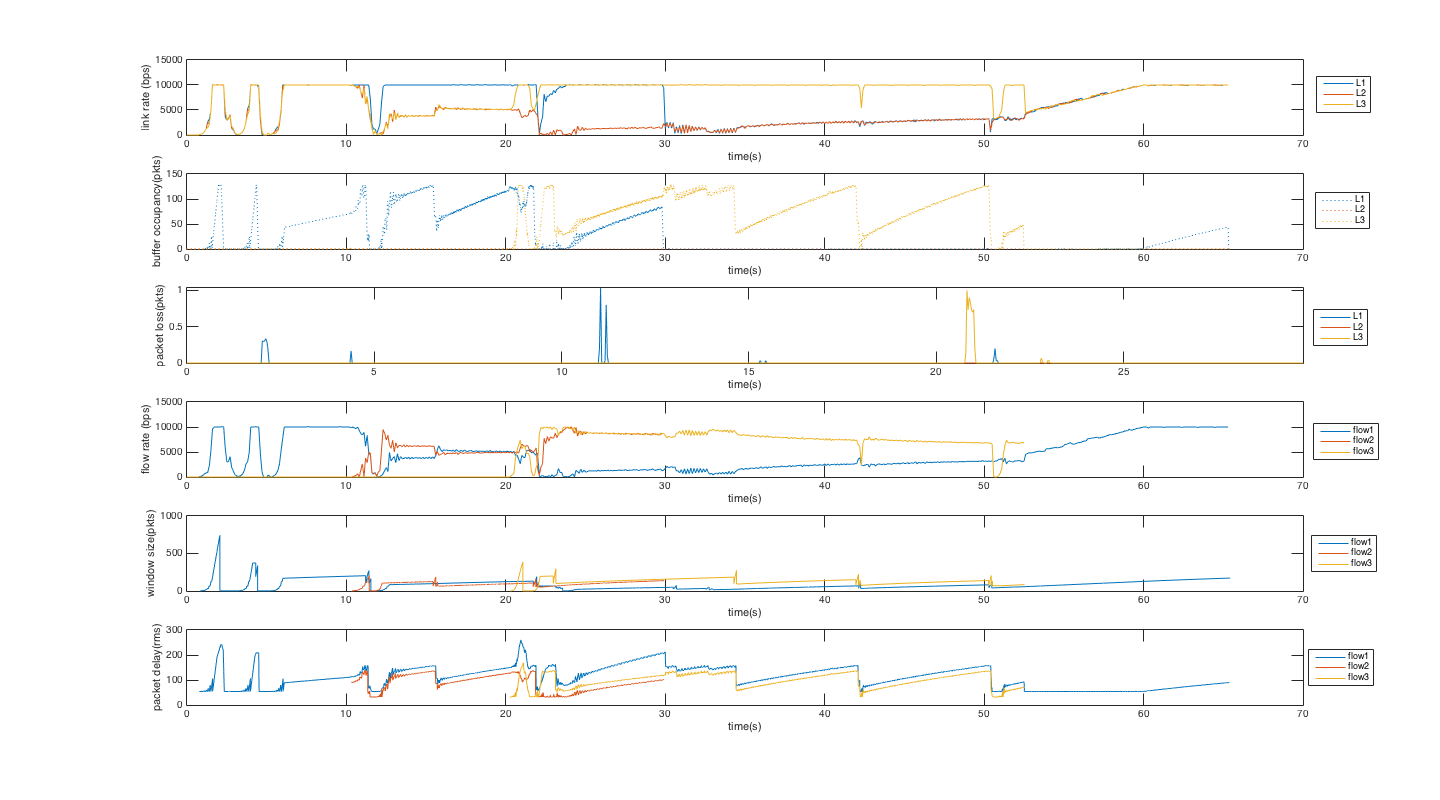
Test Case 2



RENO:



FAST:



In this case, we calculate different equilibrium point.

Calculation (basically the same as hw3, except for some parameters)

From , we can get

The corresponding continuous model of this FAST is shown below:

where

( is what the algorithm actually using, calculated by the smallest rtt)

0S~10S: Only flow 1 working. All packet are buffered at

Let

Then

Therefore

Then

Throughput rate of flow 1:

From Little’s law, Queue length of link 1:

Queue length of link 2 and link 3 is zero (from assumption).

10S~20S:

As , , , would not be met at the same time. Intuitively, we consider , , , since this is the only not trivial state for .

Therefore, we have

Let

Then

Since we have

We can get

Throughput of flow 1:

Throughput of flow 2:

Queue length of link 1:

Queue length of link 1:

Queue length of link 1:

20S~

Since , , the only not trivial steady state for is

We have

Let

Then

We have

Since

We have

The solution is

Then

Throughput of flow 1:

Throughput of flow 2:

Throughput of flow 3:

Queue length of link 1:

Queue length of link 1:

Queue length of link 1:

Then, when only Flow 1 and Flow 3 are running ,it’s approximately the same as Flow 1 and Flow 2.