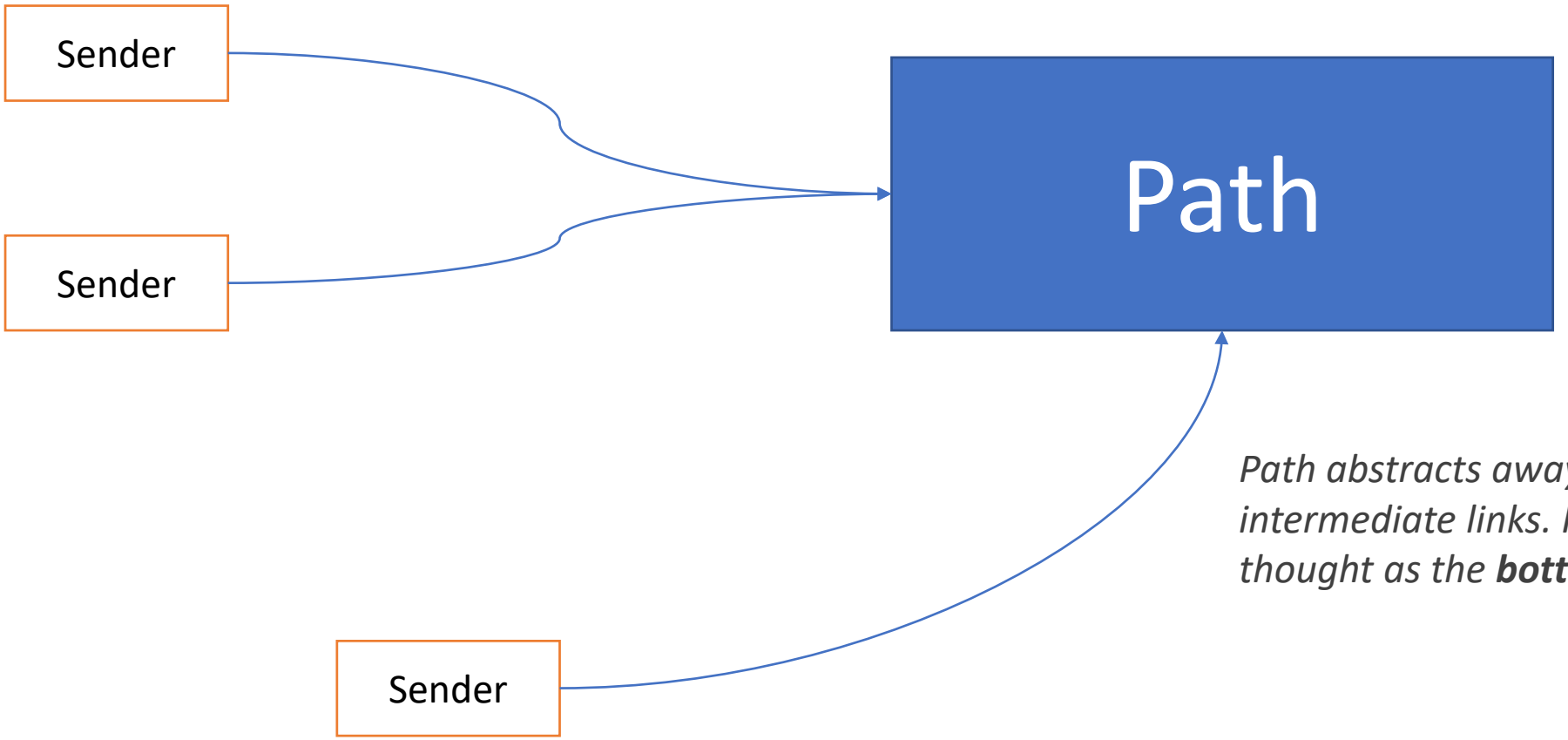
Sends in order, receives in random



*Path abstracts away the receiver, and intermediate links. It can also be thought as the **bottleneck** of a path.*

Simple control system

- A learning sender
- The bbr system has adhoc assumptions
- Learn noise
- Getting a large ttl does not reflect network congestion properly
- Beliefs that can adapt fast. Round trip time bounds given a path.
- How can we estimate the actual roundtrip time.

What does a sender know about traffic?

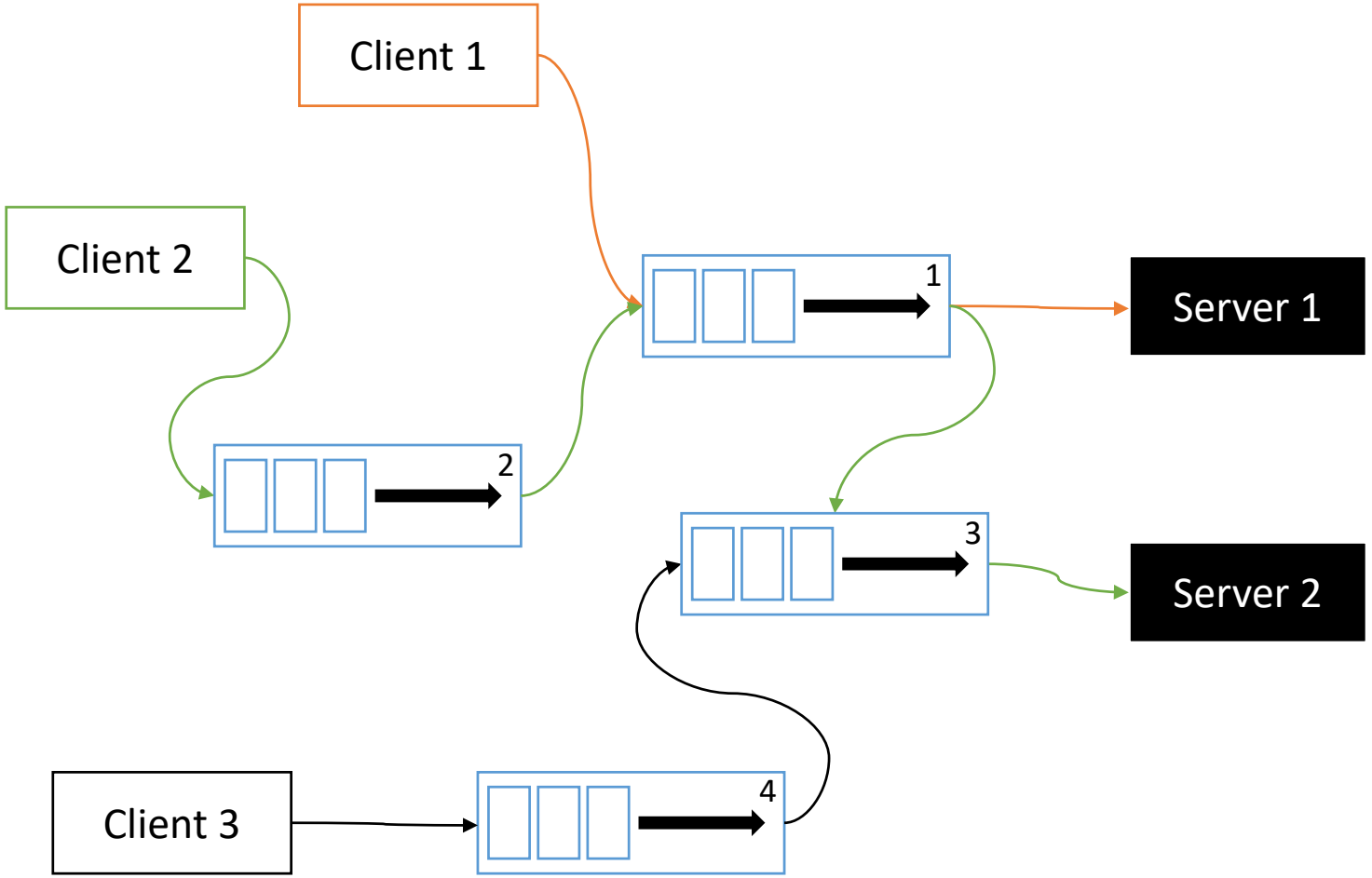
- Source
- Destination (might be our path identifier at a given period)
- Date and time
- Packet distribution over several factors
- RTT
- Neither RTProp, nor BtlBw
- IP addresses, ports of source and dest, ethernet addresses of source but not always the dest (indirect communication).
- Does it know about the ACK strategy?

What is the goal RTT (=Actual RTProp?)

- The goal RTT needs to be determined by a model which adapts fast
- Adaption rate at a given state
- Can we only think of estimating RTProp and optimizing goal RTT?
- What can RTT trend in a short time span indicate?
- Traffic changes based on the time on the day, day of the week, day of the month, or year, etc. Traffic changes based on real world events. Traffic also changes due to network topology and path selection process.

BBR

- In a time window for a connection,
  - Estimate RTProp
  - Estimate BtlBw
  - Estimate both only one at every ACK based on a check
  - Pacing delivery



Issues:

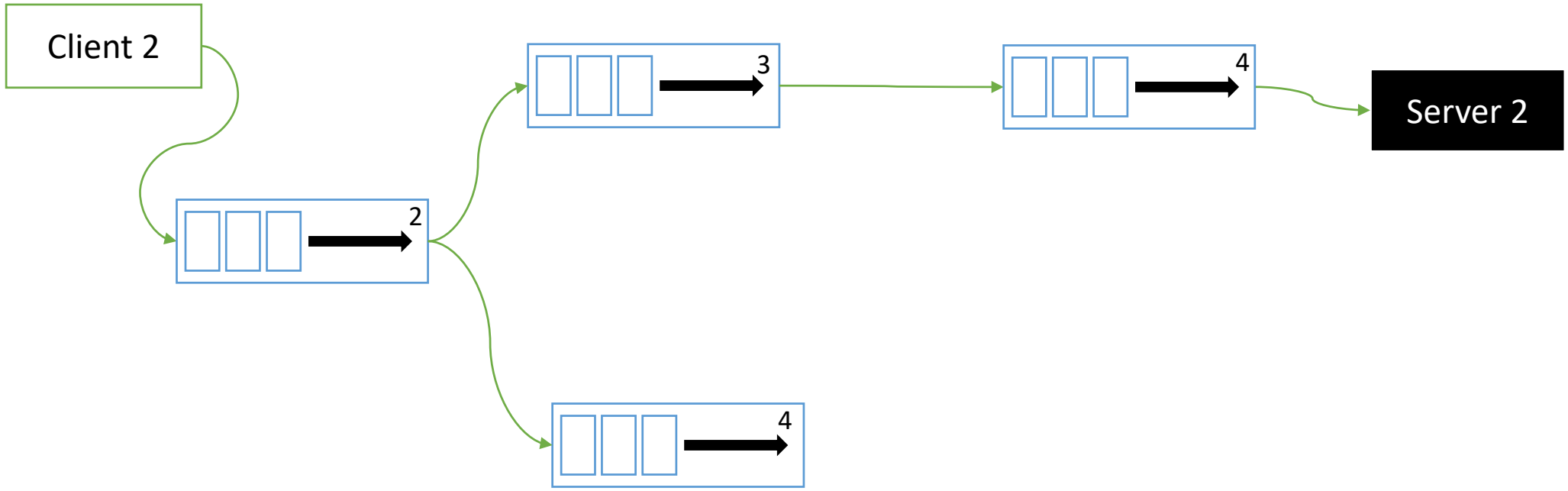
1. How packets are sent to nodes inside the network. (always broadcasting?). Routing strategy (simple, **every packet exactly knows the path it need to follow**)
2. Addressing (standard ip or numeric id)
3. If sender2's bottleneck is node 2, and client1's bottleneck is node 1, we probably cannot replace node 2 by a random TTL value assignment. If we are only interested in the path of client1, the node simulation on its path depends on the intersection of other paths.
4. Data in flight is not only packets sent by clients and servers, but also the duplicate packets.
5. Number of outstanding packets. NO ACK strategy now.
6. Throughput and latency of each node
7. Length of each queue

Node:

1. queue: hold incoming data
2. Another queue to hold data in

A path

Delay in the bluebox only.



Modeling Transmission delay between two nodes

- Two nodes connected with a wire has transmission delay
- Network object creates channels between nodes and sets the delays. The getDelay(fromNode, toNode) is useful in determining the delay from one node to another.

Path mechanism

- When a node receives some data, the delivery rate, the channel, and the queue all together works. First, the data is push to the queue, then delivery rate decides packets it can process. Next, for each outgoing packet, it removes the packet from the queue, adds a delay to the packet and schedules delivery to the next node (found from the path), and puts into the channel/pipe.
- Path has a client, and nodes. The last node acts as the server. Server has no running thread. When it receives packets, it immediately acknowledges the packets.
- Every packet has a reference to the Path object. This is useful for a node to identify next destination node for the packet.

Uncontrolled Thread-based simulation

- Each element in the network has its own thread
- All threads start at the beginning of the simulation
- Simulator collects data periodically
- All threads stop at the end of the simulation

Uncontrolled Simulation

- Each node, client, and server has their own threads and runs asynchronously.
- Time resolution: ms (mili-seconds)
- Everything works with the same resolution. This has some side-effects:
  - A client does not actually send data every ms. So, in some ms, it's possible that it sends more data than the delivery rate, which will overflow the queue. How to resolve that? Change delivery rate to per second. Need to make sure data sent over 1 second is close to the delivery rate.
  - We don't know how much each thread will block. It might be several ms. For nodes, it's better to send all the data for missing ms.

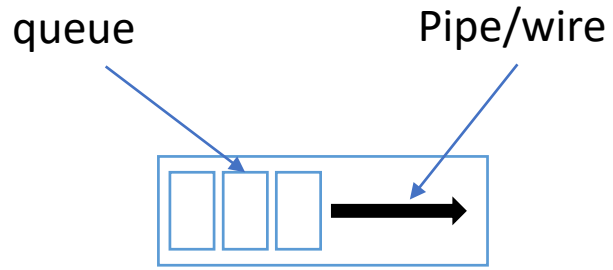
Controlled Simulation

- Simulation algorithm ensures all the nodes, clients and server runs based on synchronized time-steps. All the calculations for a timestep must be completed before the next timestep begins.

Delivery mechanism

- A previous node calls onIncomingPacket of the next node to deliver the packet.
- When a packet arrives at a node:
  - It joins the queue
  - Packets curNode and nextNode (using the path object) is updated
  - Does nothing else.

- On each simulation step:
  - Put some packets from the queue to the pipe constraint by the delivery rate of the node.
  - When a packet joins the pipe, its nodeLeaveAt is updated by the current timestep and delay of the transmission between currentNode and destinationNode.
  - Remove packets from pipe, which have nodeLeaveAt is <= timestep, and send to destination nodes



Single event queue to order the packets everywhere.

Model with Event Queue

- Resolution parameter in Nodes and Clients are not used in Event Queue model
- EventQueue with actual time will create **time gaps** in creating and routing packets, because of processing of events. So, instead, we create simulation timestep which increases by 1 after each step. The resolution parameter says whether it’s a ms/mcs/ns. A lot of other parameters are defined in ms/s. So, we need to adjust their calculations based on simulation timestep unit.
- Heap is sometime reconstructed and events with the same time can be reordered. So, we need to serialize events by adding microseconds.

Transmission delay in MS

Transmission delay is the amount of time required to push one BYTE to a channel: A 20 B packet requires:

20 * transmissionDelayPerByte amount of time.

If **transmissionDelayPerByte** is 0.0001ms, it would take 0.002 ms or 2 micro seconds to push the packet into a channel.

Where there will be no queuing

Let’s say our queue size is 5 and 5 packets arrived at time 1000Here are the events that will happen:

Arrive p1

Channel p1 (because as queue was empty, p1 will be pushed to the channel at 1000)

Arrive p2

Channel p2

Arri…

Because even if we had [A1, A2, A3, A4, A5] in the event queue at time 1000. Each of the arrival event will be put before the pending arrival events. So, queue will never have more than 1 item.

Experimental plan

- Analysis RTT change over n-RTT time
- Some power calculation which considers both expected throughput and RTT
- Try to stress the network and see what would be the highest throughput.
- Then try to achieve the best RTT
- Then try to optimize a trade-off function.

Visualisation

- Node summary every 100ms.
- All nodes like a barchart
- Client summary every 100ms and put on the nodes.

Experimental Setups

Client 1 DeliveryR	Client 1 Outs. P	Client 2 DeliveryR	Client 2 Outs. P	
3000	10	3000	10	done
3000	100	3000	100	done
2000	10	2000	10	
2000	100	2000	100	
1500	10	3000	10	
1500	10	100	10	
1500	10	500	10	

Impact of delivery rate (node 2 at 2000/s)

id	Client 1 DeliveryR	Client 1 Outs. P	Client 2 DeliveryR	Client 2 Outs. P	Comments on queue and rtt
	1500	100	1500	100	Outstanding packets (30, 40), rtt(60, 70), queue (110, 120)
	1500	100	1000	100	Outstanding packets (30, 40), rtt(60, 70), queue (110, 120)
	1500	100	500	100	Outstanding packets (1, 2), rtt(2, 2.5), queue (2, 3)
	1000	100	1500	100	Outstanding packets (40, 50), rtt(50), queue (80)
	1000	100	1000	100	Outstanding packets (1, 2), rtt(1.5, 2.5), queue (1, 2)
	1000	100	500	100	Outstanding packets (1), rtt(1.3), queue (0)
	1000	100	1010	100	Outstanding packets (1, 2), rtt(1.5, 2.5), queue (1, 2)
	1000	100	1100	100	Outstanding packets (16, 20), rtt(16,20), queue (30)
	500	100	1500	100	Outstanding packets (1, 3), rtt(1.5, 2.5), queue (1, 3)
	500	100	1000	100	
	500	100	2500	100	Outstanding packets (70, 80), rtt(50, 60), queue (80)

When the combined delivery rate (delay between packets) of the clients is higher than the bottleneck's channeling rate, convergence is very slow. This is because they initially inflate the queues.

Impact of delivery rate (node 2 at 500/s)

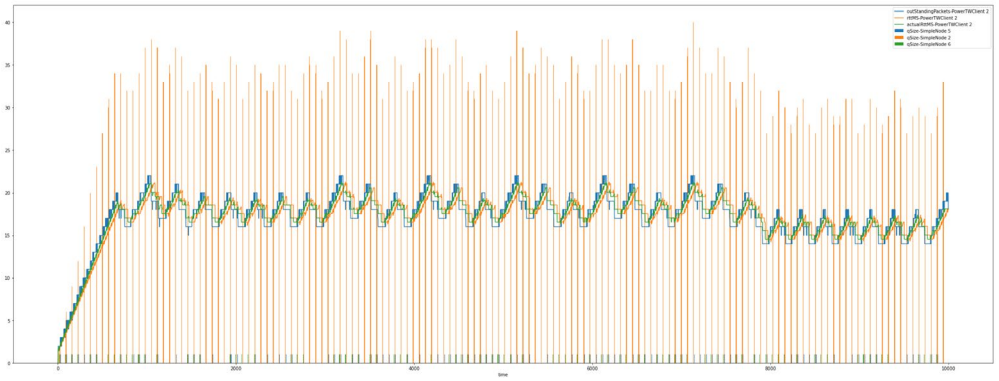
id	Client 1 DeliveryR	Client 1 Outs. P	Client 2 DeliveryR	Client 2 Outs. P	Comments on queue and rtt
15-100-15-100	1500	100	1500	100	Outstanding packets (30, 40), rtt(60, 70), queue (110, 120)
15-100-10-100	1500	100	1000	100	Outstanding packets (30, 40), rtt(60, 70), queue (110, 120)
	1500	100	500	100	Outstanding packets (1, 2), rtt(2, 2.5), queue (2, 3)
	1000	100	1500	100	
	1000	100	1000	100	
	1000	100	500	100	
	1500	10	1000	10	
	1500	10	500	10	
	1500	10	300	10	
	500	100	1500	100	
	500	100	1000	100	
	500	100	2500	100	

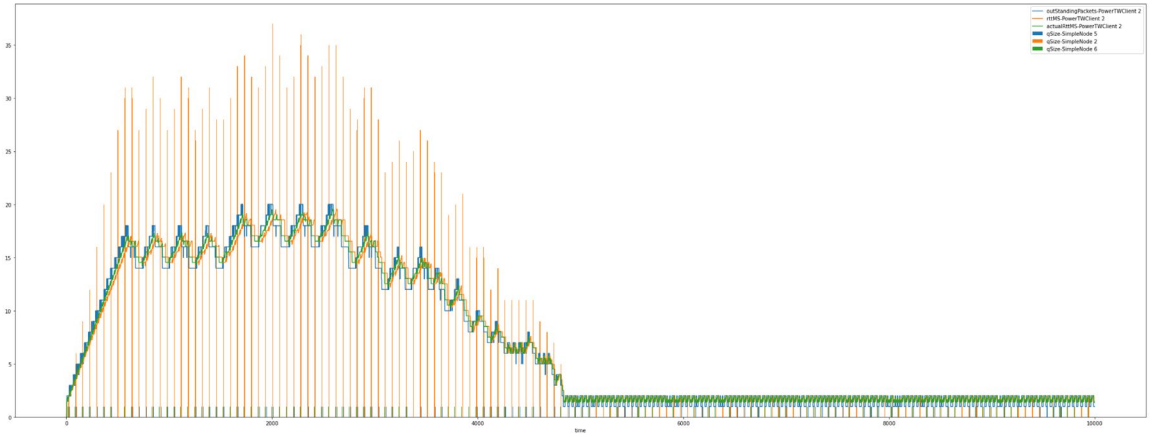
Impact of exploration when power increases.

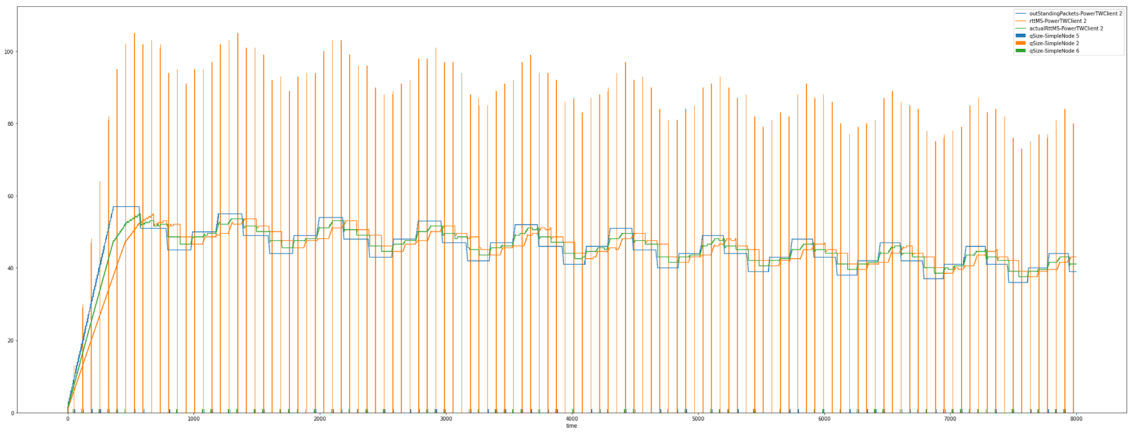
id	Client 1 DeliveryR	Client 1 Outs. P	Client 2 DeliveryR	Client 2 Outs. P	Comments on queue and rtt
	1500	100	1500	100	
	1500	100	1000	100	
	1500	100	500	100	
	1000	100	1500	100	Outstanding packets (20), rtt(20), queue (20)
	1000	100	1000	100	
	1000	100	500	100	
	1000	100	1010	100	
	1000	100	1100	100	Outstanding packets (1, 2), rtt(2, 2.5), queue (2, 3)
	500	100	1500	100	
	500	100	1000	100	
	500	100	2500	100	

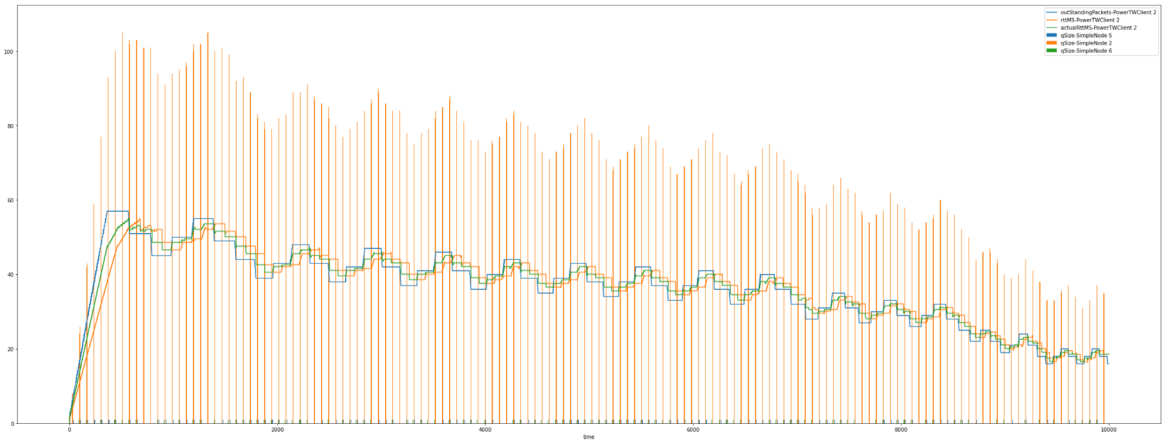
Impact of exploration when power increases. (long network)

Impact of exploration when power increases.





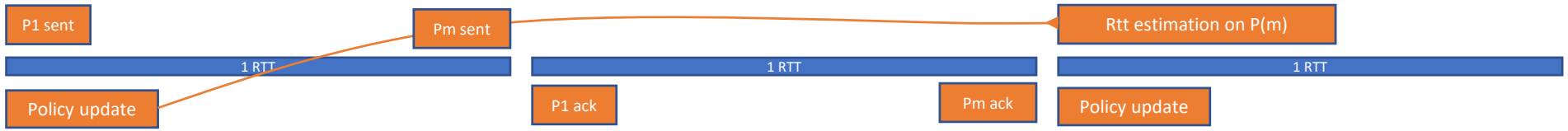




No exploration

Exploration : 0.2

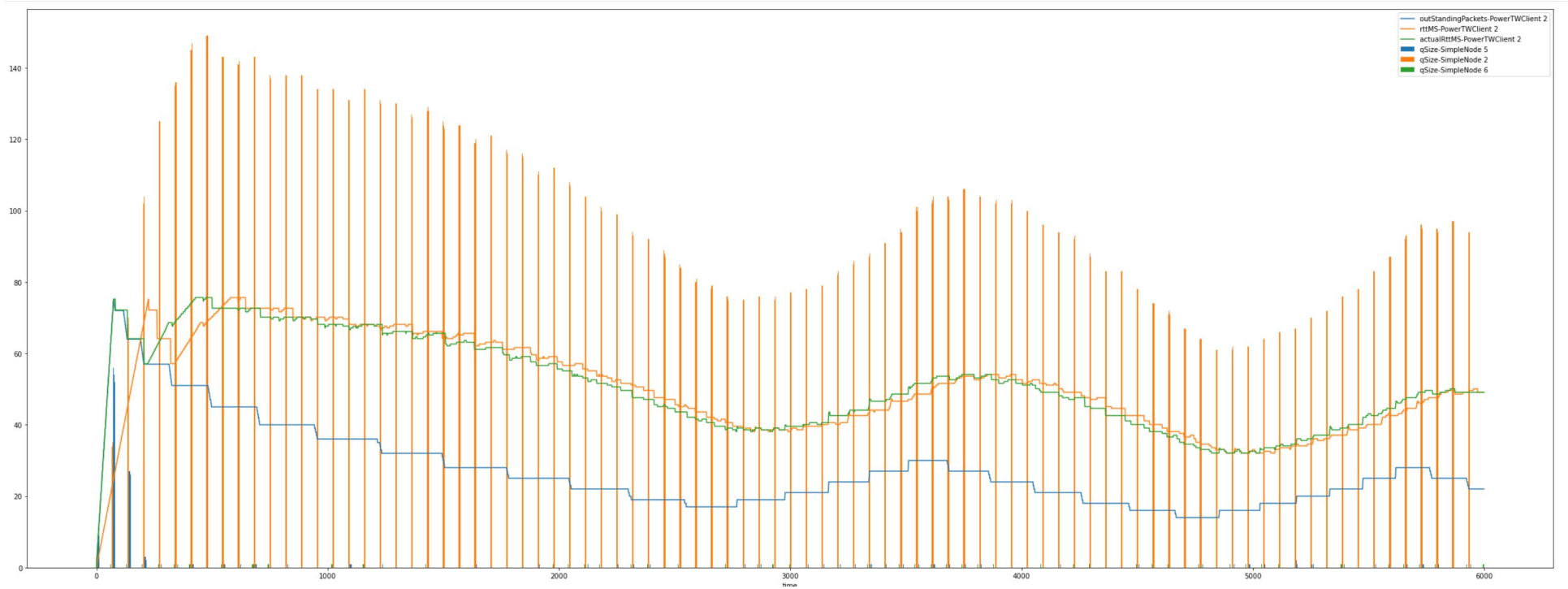
RTT estimation and policy update cycles



Whenever we update the policy, the impact on rtt can only be reliably measured after 2 currentRTT times. But there will still be impact of previous queue sizes before the policy updates.

Interesting observations for 100 max outstanding packets with delivery rate 1.5k both clients

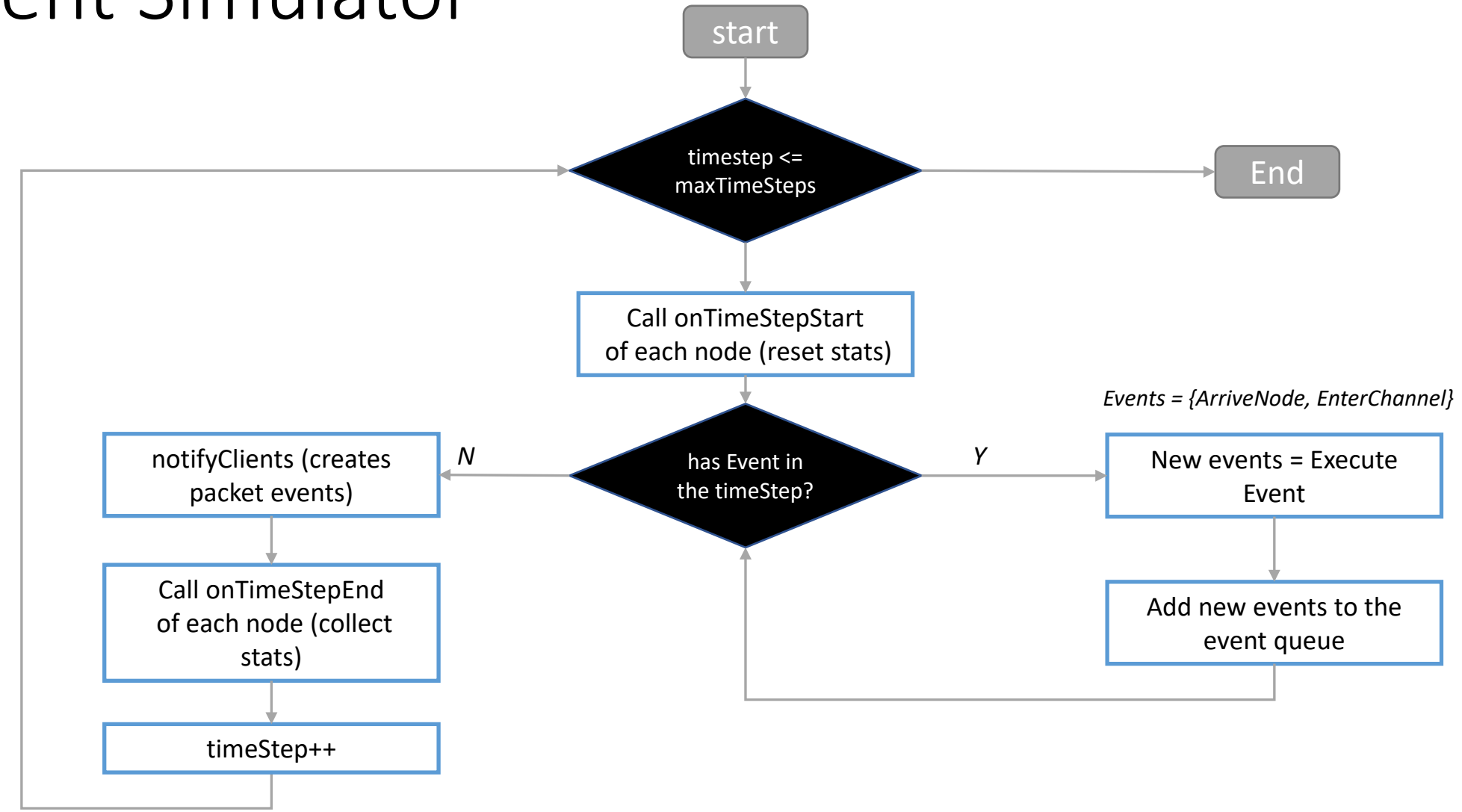
- The client does not try to optimize after a lower boundary
- We need to add some exploration mechanism
- What to do when rtt does not change?



Client ideas

- Find optimal rtt and throughput in the beginning. And try to find a trade off.
- RL #1: state (an MDP on S = (rtt, max outstanding, curr outstanding) A = amount of increase/decrease = {+-10%, 20 %) = discrete values.
- RL #2: state (S = (a short history of power, rtt, max outstanding, curr outstanding)
- Q(s,a) = goal rtt – curRtt.
- Exploration, e = 0.1 or a function. E can be reset.
- Power trend over several rtts (right now 1 rtt).

Event Simulator



Algo 1. policy cycle

1. Estimate rtt
2. Update policy
3. Wait 2rtt time
4. Go back to 1

PowerClient - policy

Power = outstanding_packets / rtt^2

If power 2rtt before is less than current power (increase):
    if exploit:
        increase max_*outstanding_packets by 20%*
        return
Decrease max_*outstanding_packets by 20%*
If max_*outstanding_packets < minimum_threshold:*
    max_*outstanding_packets = minimum_threshold*

return