Managing Fairness and Application Performance with Active Queue Management in DOCSIS-based Cable Networks

Presented at

ACM SIGCOMM Capacity Sharing Workshop (CSWS 2014)

Jim Martin, Mike Westall
School of Computing
Clemson University
Clemson, SC
jim.martin@cs.clemson.edu

Students:
Gongbing Hong
hgb.bus@gmail.com

An extended version of the paper and AQM source code are available at:
http://people.cs.clemson.edu/~jmarty/AQM/AQMPaper.html

Agenda

- Summary of the research and the contribution
- Motivations
- Background
- Methodology
- Results
- Conclusions and next steps
- Acknowledgements

Summary of the Research and the Contribution

- Using an ns2-based simulation model of DOCSIS 3.0, we address the following questions
  - How effectively do CoDel and PIE support fairness and application performance in realistic cable scenarios?
  - Are there issues when AQM interacts with tiered service levels?
  - How effectively do the schemes isolate responsive traffic from unresponsive flows?
- Contribution:
  - Better understand delay-based AQM when considering tiered service levels, workloads that include HTTP-based adaptive streaming (HAS), and DOCSIS cable environments

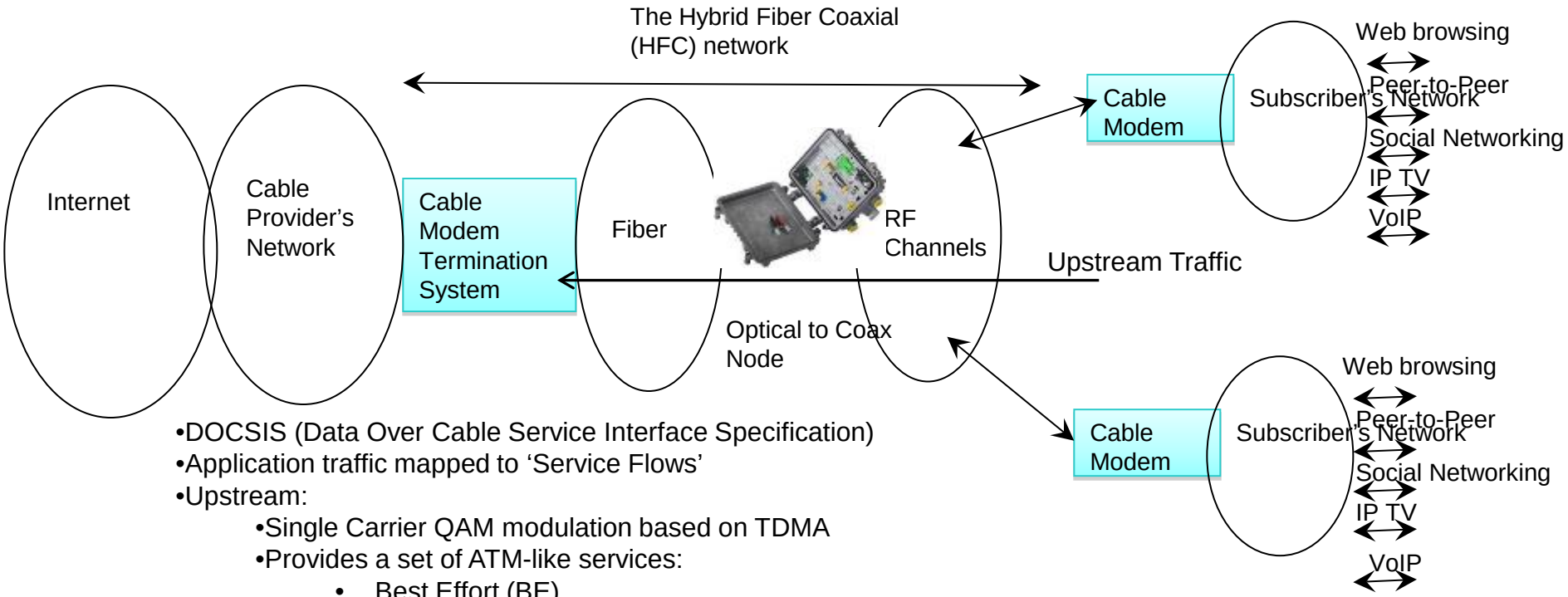
Motivations

- The field of AQM is well established however….why does bufferbloat still exist?
- Progress …we are moving towards large scale deployments of a standard AQM!
- To the best of our knowledge, delay-based AQM has not been studied stood in cable environments with tiered service levels or HAS workloads.

Background : AQM

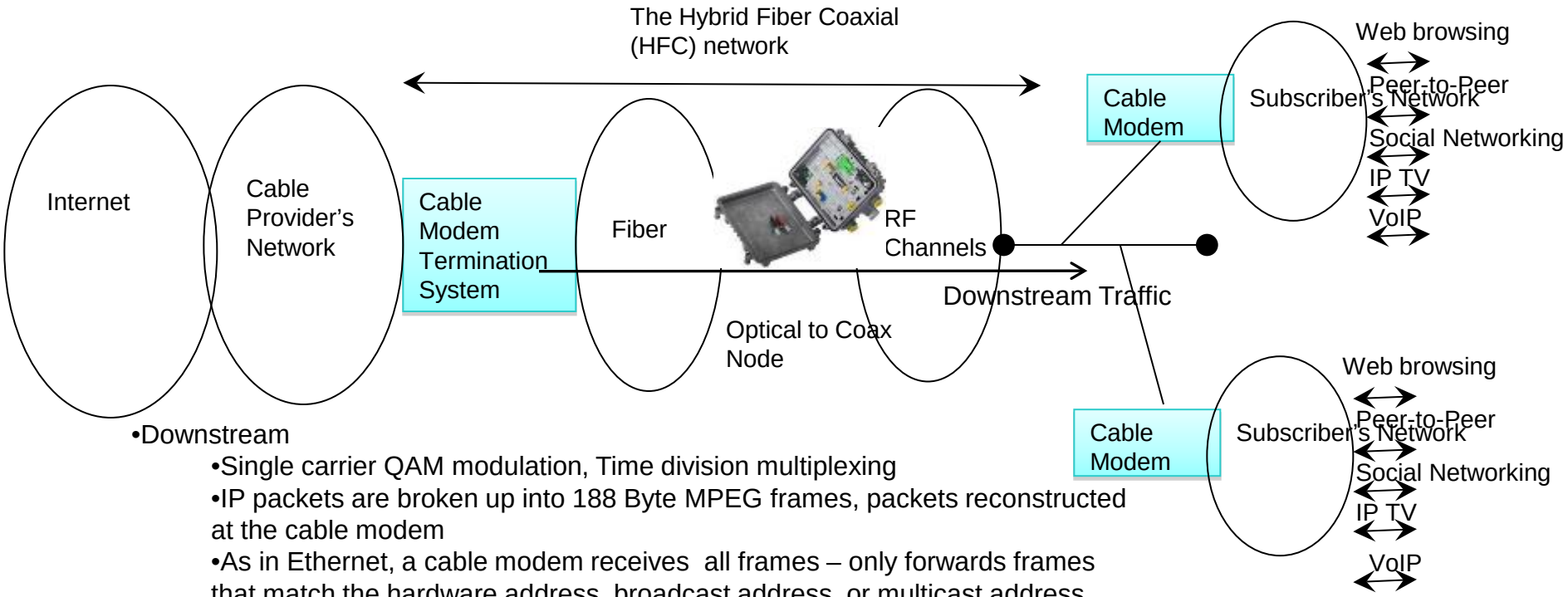
- It has been shown that RED is sensitive to traffic loads and parameter settings
- Renewed AQM manifesto (IETF Recommendations Regarding Active Queue Management draft-ietf-aqm-recommendation-08)
  - Deployments should use AQM, should not require tuning, should respond to measured congestion such that outcomes are not sensitive to application packet size
- Led to :
  - Relook at Adaptive RED : adapts the RED *maxp* parameter to track a target queue level
  - Two new delay-based AQMs: Controlled Delay (CoDel) and Proportional Integral Controller (PIE).
    - Both introduce two parameters:
      - Target_delay : sets a statistical target queue delay bound
      - Control_interval: defines the timescale of control
    - DOCSIS 3.1 requires cable modems to support PIE (recommended for DOCSIS 3.0); Both DOCSIS 3.1 and 3.0 recommend a published AQM

Background : DOCSIS-Based Cable



- DOCSIS (Data Over Cable Service Interface Specification)
- Application traffic mapped to 'Service Flows'
- Upstream:
  - Single Carrier QAM modulation based on TDMA
  - Provides a set of ATM-like services:
    - Best Effort (BE)
    - Unsolicited Grant Service (UGS)
    - Real-time Polling Service (RTPS)
    - Non-real-time Polling Service (NRTPS)
  - An upstream scheduler computes the allocation for the next 'map time'….regularly broadcasts a MAP message with grants to all Cable Modems

Background : DOCSIS-Based Cable



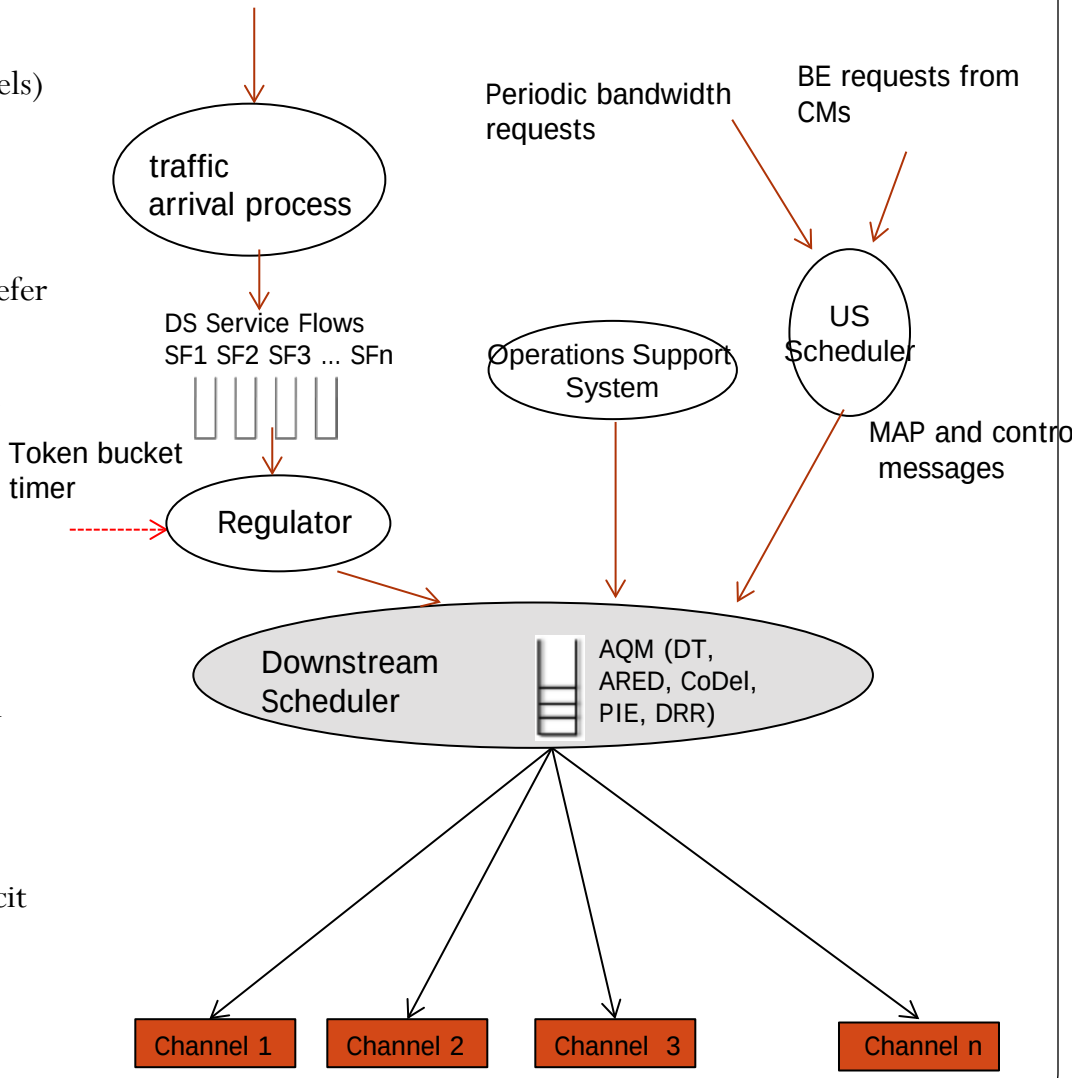
- Downstream
  - Single carrier QAM modulation, Time division multiplexing
  - IP packets are broken up into 188 Byte MPEG frames, packets reconstructed at the cable modem
  - As in Ethernet, a cable modem receives all frames – only forwards frames that match the hardware address, broadcast address, or multicast address
- Support for multiple channels
  - Cable Modems equipped with multiple downstream tuners
  - Downstream services flows are assigned to a bonding group
  - A bonding group is an abstraction that represents the group of channels that are available to a service flow.
  - For downstream, the scheduler can allocate bandwidth to service flows from any channel that is in the bonding group

Background : DOCSIS Standards

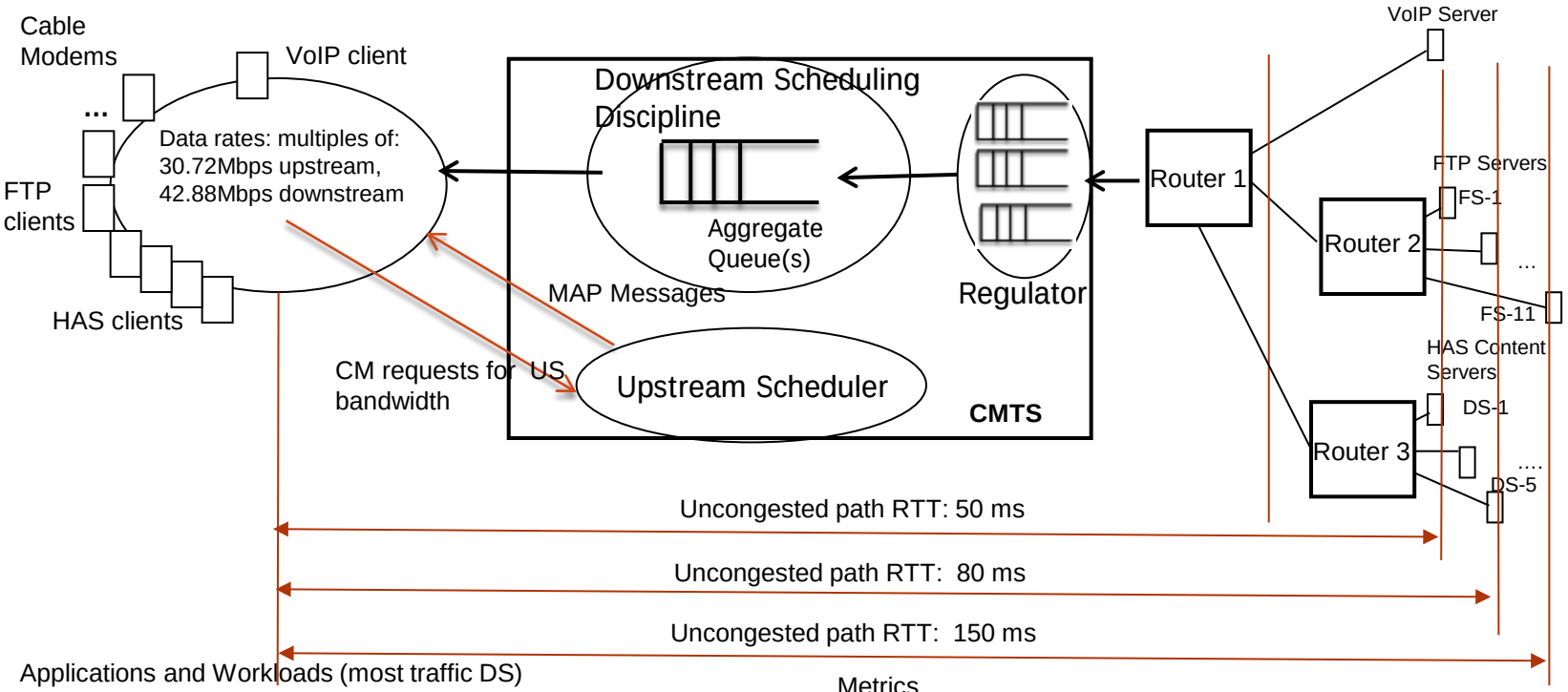
- DOCSIS 1.0: base version.
  - About 30Mbps DS, 5 Mbps US
- DOCSIS 1.1/2.0:.
  - Added QoS capabilities.
  - About 42 Mbps DS, 30 Mbps US
- DOCSIS 3.0: channel bonding
  - Any number of 6MHz DS channels can be combined (4 to 8 common)
  - Up to 4 US channels can be combined
- DOCSIS 3.1:
  - Spectrum rebanding (24 to 192 MHz channels)
  - OFDM for DS, OFDMA for US
  - Adaptive modulation and coding through a set of standard profiles

Methodology: CMTS System Model

- We consider scenarios involving 1 DS channel (an update to our extended paper considers 4 DS channels)
- We consider scenarios that include use of a token bucket rate shaper/limiter
  - Includes cases when different service flows are provisioned with different 'service rates' – we refer to this as tiered service levels
- We assume each subscriber operates one application
- We consider competing services flows will interact with different servers located at possibly different geographic locations.
  - We consider scenarios where services flows experience different uncongested path RTTs
- Experiments are designed such that the downstream cable network is the single bottleneck AND this is where one of the following queue management is performed:
  - Drop Tail, Adaptive RED, CoDel, PIE, and Deficit RR



Methodology



Applications and Workloads (most traffic DS)

- FTP: varied the number
- VoIP: server sends G.711 traffic to the client (always on), compute the R-Value based on observed latency, jitter, loss
- HTTP-based Adaptive Streaming (HAS) :
  - Server : HTTP server that receives requests for a specific segment of content from one of a set of available bitrate representations.
  - Client: requests segments as needed, adapting the requested video quality to minimize the frequency of buffer stalls while maximizing video playback quality

Metrics

- Application level:
  - FTP : throughput, TCP RTT, TCP loss rate
  - VoIP: latency, loss, R-Value
  - HAS: average video play back rate and frequency of video adaptation
- System level (Fairness measures): (see next slide)
  - Jain's fairness index:
  - Min-Max ratio
  - Allocation error:

Methodology

- We define the normalized throughput for the *i'th* user among *n* users competing for channel capacity as
  - $x_i = {}^{T_i}/_{r_i}$ where T_i is the achieved throughput of the *i'th* flow and r_i is the expected outcome based on a max-min criteria
- Jain's Fairness Index is defined as:
  - $JFI = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n \sum_{i=1}^{n} x_i^2}, x_i \geq 0$
- The min-max ratio is simply : $\frac{Min\{x_i\}}{Max\{x_i\}}$.
- The allocation error is the ratio of the difference between the average achieved throughput of *n* similar flows and r (the expected outcome) to r : $\left[{}^{[\sum_{i=1}^{n} T_i]}/_{n}\right] - {}^{r}/_{r}$

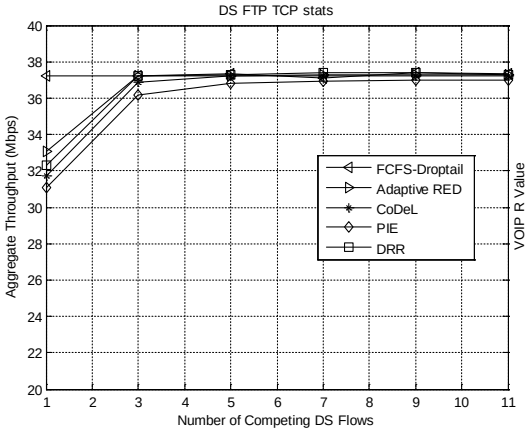
Methodology

Experiment ID	Summary
EXP1	All flows not limited by service rates, vary the number of competing DS FTP flows
EXP2	Same as EXP1 except add five HAS flows
EXP3	Same as EXP1 except: 1)Set all flow services rates to 6 Mbps (refer to these flows as Tier 1 flows); 2)Add one additional competing FTP flow with a service rate of 12 Mbps that is active only during the time 500 – 1500 seconds. Refer to this flow as a Tier 2 flow.
EXP4	Same as EXP1 except add a 12 Mbps DS UDP flow (starts at 500 seconds, stops at 1500 seconds)

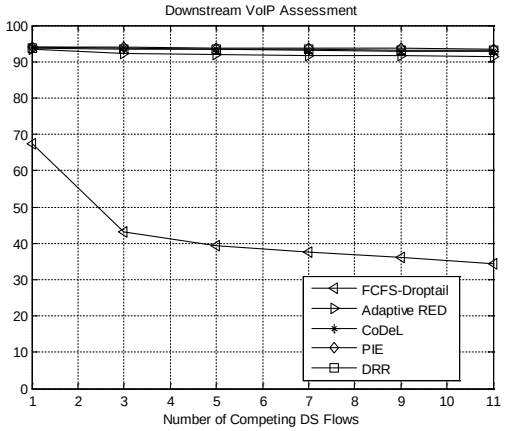
More details ……

- Simulation run for 2000 seconds
- All results described use the staggered (varying) path RTT
- The max TCP window is set to 10000 packets
- The buffer capacity of the bottleneck link is 2048 packets
    - For DRR that uses multiple queues, each per flow queue set with a capacity of 2048/16
    - The buffer capacity of all other links is 4096

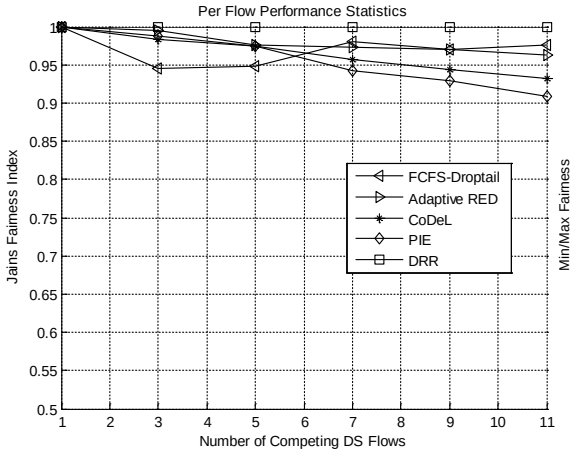
Results : EXP1



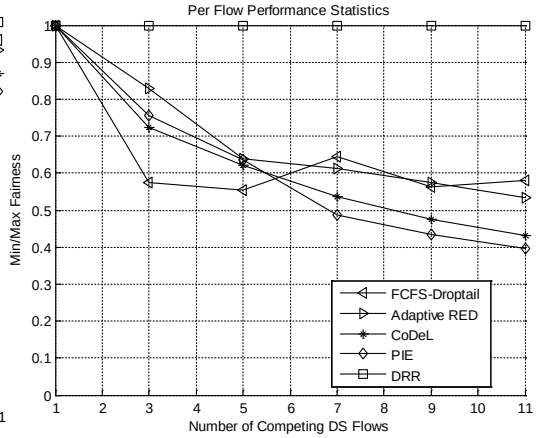
a. Aggregate Throughput



b. VoIP R-Value



c. Jain's Fairness Index



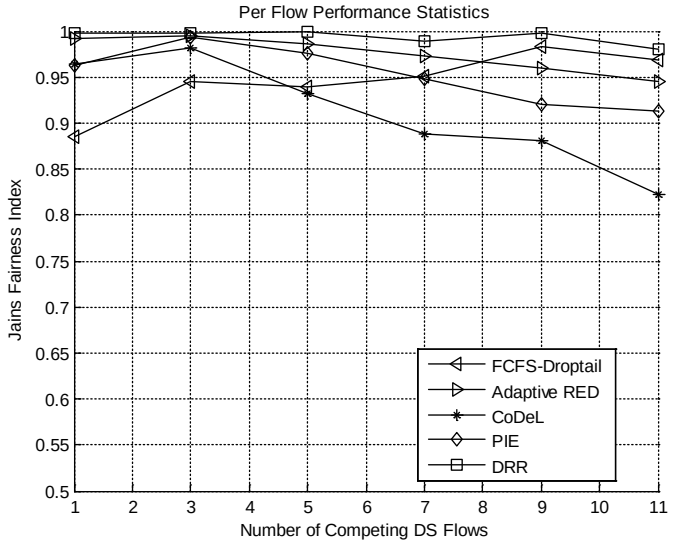
d. Min-Max Ratio

Results:

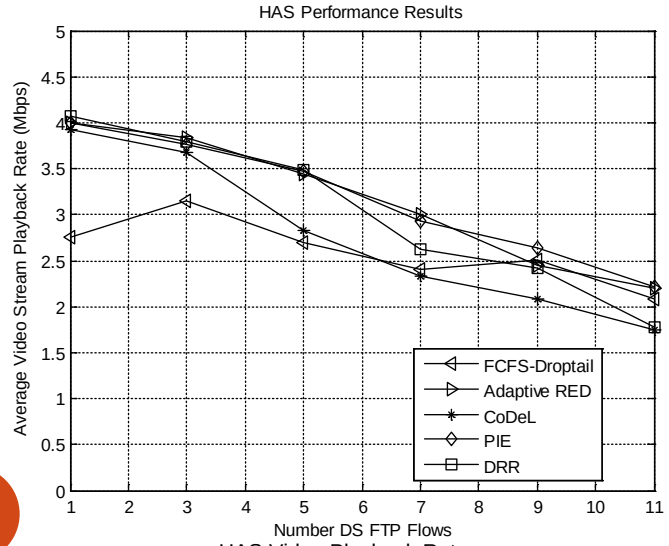
- Fig. a: Each AQM leads to slightly different aggregate throughputs…converge at high levels of congestion.
- Fig. c/d: As expected, DRR achieves a correct max-min allocation, Drop-tail has synchronization issues, PIE/CoDel seems the most sensitive to TCP/RTT unfairness.
- Fig. b: delay-based AQM very effective at providing isolation

Results : EXP2

#FTPs	1	3	5	7	9	11
FTP Flow	17	5.67	3.8	3.17	2.71	2.38
HAS Flow	4.2	4.2	3.8	3.17	2.71	2.38



a. JFI



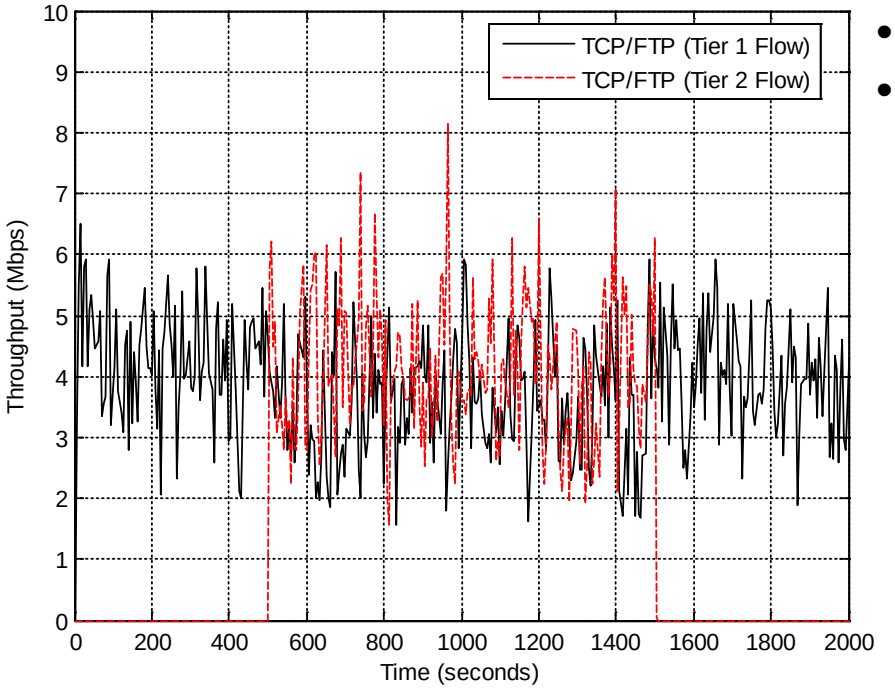
c. HAS Video Playback Rate

Results:

- Fig. a: We assume HAS flows have a demand of 4.2 Mbps, leading to an expected max-min allocation shown above.
  - Outcomes more variable than EXP1
  - DRR is not exactly max-min
  - Allocation error suggests that HAS allocation is less than max-min although the error moves from on the order of 25% to 5% as the load increases
- Fig. c: Suggests a similar, linear decrease in video quality
  - Further analysis is required to make sense of the other HAS metrics

Results : EXP3

During time interval 500 – 1500 seconds:
Tier 1 Flow mean/std (Mbps): 3.54 / 0.39
Tier 2 Flow mean/std (Mbps): 4.10 / 0.46



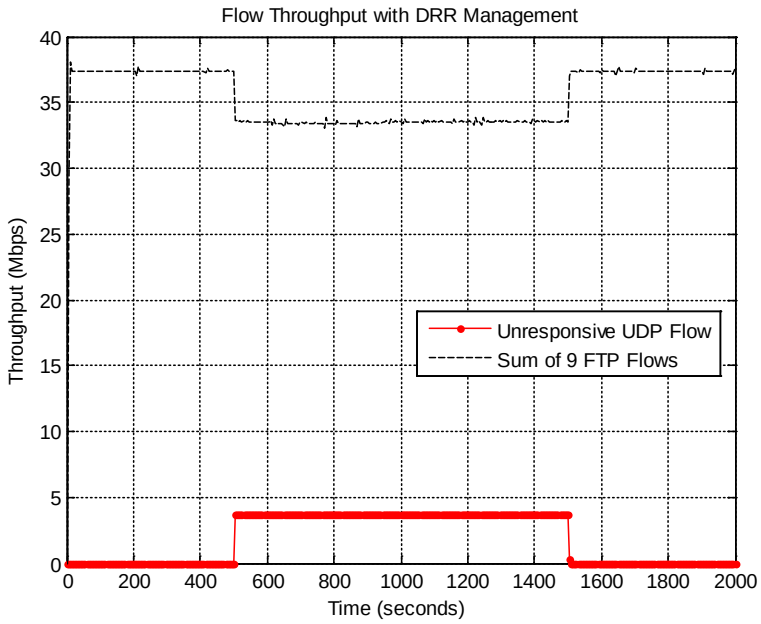
CoDel Result: One Tier 1 flow (out of 9) and the Tier 2 Flow Compete (similar path delays)

Results:

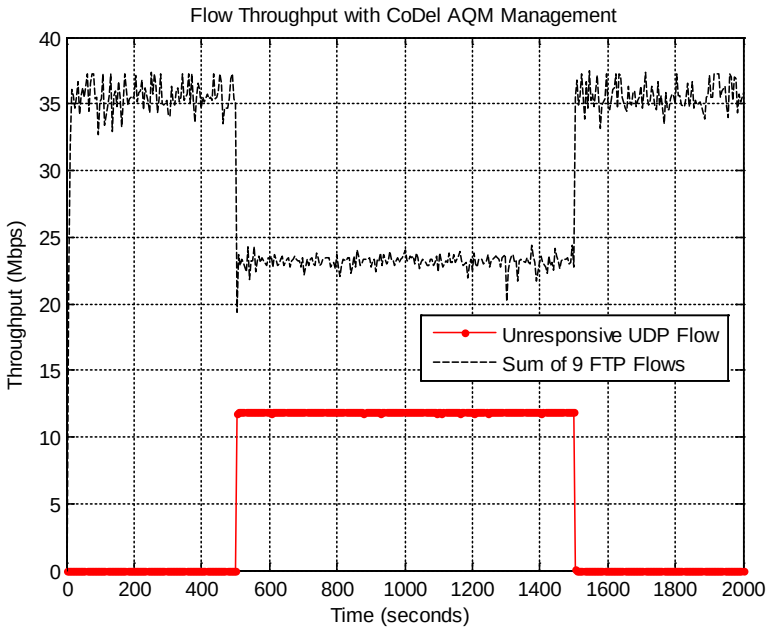
- We assume non-weighted max-min is the intended outcome
- Max-min is 3.8 Mbps (38 Mbps /10 flows)
- Tier 2 achieves higher allocation in all cases except for PIE and CoDel at the highest load level

	9 FTPs	9 FTPs	11 FTPs	11 FTPs
	Tier 1	Tier 2	Tier 1	Tier 2
DT	-14.2105	137.1053	-11.6719	154.5741
ARED	-7.36842	2.894737	-5.99369	6.624606
CoDel	-6.84211	7.894737	-8.51735	-5.99369
PIE	-7.63158	-4.73684	-9.46372	-13.2492
DRR	-1.57895	-2.10526	-1.57729	-2.2082

Results : EXP4



DRR



CoDel

Conclusions and Next Steps

- Motivating Questions:
  - How effectively do CoDel and PIE support fairness and application performance in realistic cable scenarios?
    - PIE/CoDel seems the most sensitive to TCP/RTT unfairness
    - delay-based AQM very effective at providing isolation between high BW flows and low BW latency sensitive flows
  - Are there issues when AQM interacts with tiered service levels?
    - Allocation error suggests that HAS allocation is less than max-min although the error moves from on the order of 25% to 5% as the load increases
  - How effectively do the schemes isolate responsive traffic from unresponsive flows?
    - As one would expect, single queue AQM is not able to manage unresponsive flows
- Further analysis is required, but there appears to be a tradeoff between maintaining a low delay and fairness
- Moving Forward:
  - Include scenarios involving higher speeds….we are developing a technique that allows uncapped (i.e., no token bucket rate limiting) except during times of congestion
  - We are develop a two-queue AQM method that can better address unresponsive traffic

Acknowledgements

- Thanks to my team in the networking lab, especially Gongbing Hong and Yunhui Fu.
- Thanks to Greg White at CableLabs who developed the tail-drop CoDel ns2 code.
- Thanks to Rong Pan (and others at Cisco) who developed the PIE ns2 code.
- Thanks to the anonymous reviewers- very helpful comments and feedback.
- Thanks to current and prior sponsors of our work in DOCSIS including CableLabs, Cisco, Comcast, Cox Communications and Huawei.

Appendix

- Step 1: identify the communities
- Step 2: identify common goals and objectives
- Step 3: coordinated plans

On packet arrival to a system where all channels are busy:

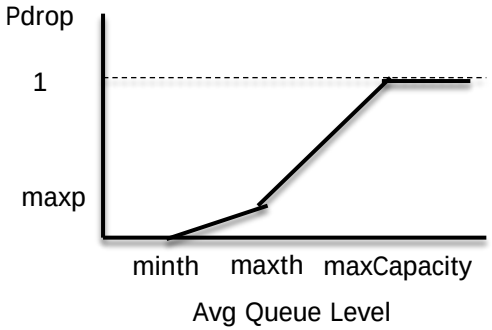
```
dropPacketFlag = FALSE;
curTime = getCurrentTime();
avgQ = (1-Wq)*avgQ + Wq*q.curSize();

if (minth  <=avgQ <=maxCapacity) {
  count++
  Pdrop = computeDropProbability(avgQ,count)
  dropPacketFlag =  dropDecision(Pdrop)
}  else if (avgQ >=maxCapacity)
  dropPacketFlag = TRUE;
else
  count = -1;

if (dropPacketFlag == TRUE) {
  q.drop(pkt)
  count=0
} else if maxth <= avgQ
  q.drop(pkt)
else
  q.enqueue(pkt);


//Every Tupdate seconds:
if (curTime – lastUpdateTime > Tupdate)
  lastUpdateTime = curTime;
  If (avg > target) and maxp <= 0.5
    maxp+=alpha
  else if (avg < target and maxp >= 0.01)
    maxp = beta*maxp
}
```

Configured parameters:

- maxCapacity: maximum queue capacity (2048 packets)
- Wq: queue average time constant (0.002 default)
- minth: minimum threshold (maxCapacity / 20 packets)
- maxth: maximum threshold (maxCapacity /2 packets)
- Tupdate: frequency of adaptation (0.50 seconds)
- target: target queue level ((minth + (minth+maxth) / 2) packets)
- alpha: maxp / 4
- beta: 0.90
- maxp : initial setting 0.10

state variables:

- avgQ: averaged queue size
- count: packets since last drop
- Pdrop : drop probability
- maxp : maximum value for Pdrop



Adaptive RED

On packet arrival to a system where all channels are busy:

```
curTime = getCurrentTime();
pkt->timestamp=curTime;

if (q.curSize() + 1 > maxCapacity) {
  q.drop(pkt)
  return;
}

dropFlag = NO_DROP;
if (dropping) {
  if (drop_now) {
    dropFlag  = DROP;
    drop_now_ = 0;
  } else if (drop_next == 0.0) {
     count++;
     drop_next =  curTime +  interval / sqrt(count);
  } else if (curTime > drop_next) {
     dropFlag = DROP;
  }
}
If (dropFlag== DROP) {
  q.drop(pkt);
  count++;
  drop_next_ =curTime +  interval / sqrt(count);
}
else
   q.enqueue(pkt);
```

When a channel is available :

```
curTime = getCurrentTime();
pkt = q.dequeue();
d_exp = curTime - pkt->timestamp;
if (dropping) {
  if (d_exp < target)
     switchToNotDropping() ;
} else  {
  if (d_exp > target) {
    if (first_above_time == 0) {
      first_above_time  = curTime + interval;
    } else if (curTime >= first_above_time) {
       switchToDropping();
    }
  } else {
     first_above_time  = 0;
  }
}
```

Configured parameters:

- maxCapacity: maximum queue capacity (2048 packets)
- target : target queue delay (0.020 seconds)
- interval : packet latency must exceed the target for an interval amount of time before moving to dropping state (0.100)

state variables:

- dropping: flag indicating if in dropping state
- drop_next: The time for the next scheduled drop
- count : The number of drops since entering the last drop state

On packet arrival to a system where all channels are busy:

```
curTime = getCurrentTime();
if (curTime – lastUpdateTime > Tupdate) {
  cur_delay= q.curSizel() / departure_rate;

  Pdrop = Pdrop+ alpha*(cur_delay – target_delay) +
                  beta*(cur_delay-prev_delay)

  if ( (cur_delay < (0.5* target_delay)) &&
      (prev_delay  < (0.5* target_delay)) &&
      ( Pdrop == 0) ) {
    dq_count = -1;
    avg_dq_rate = 0.0;
    burst_allowance = max_burst;
  }
  prev_delay = cur_delay;
  lastUpdateTime = curTime
}

dropPacketFlag =  dropDecision(Pdrop)

if (dropPacketFlag == TRUE) {
  if (burst_allowance > 0)
    q.enqueue(pkt);
  else
    q.drop(pkt);
} else
  q.enqueue(pkt);
}
```

Configured parameters:

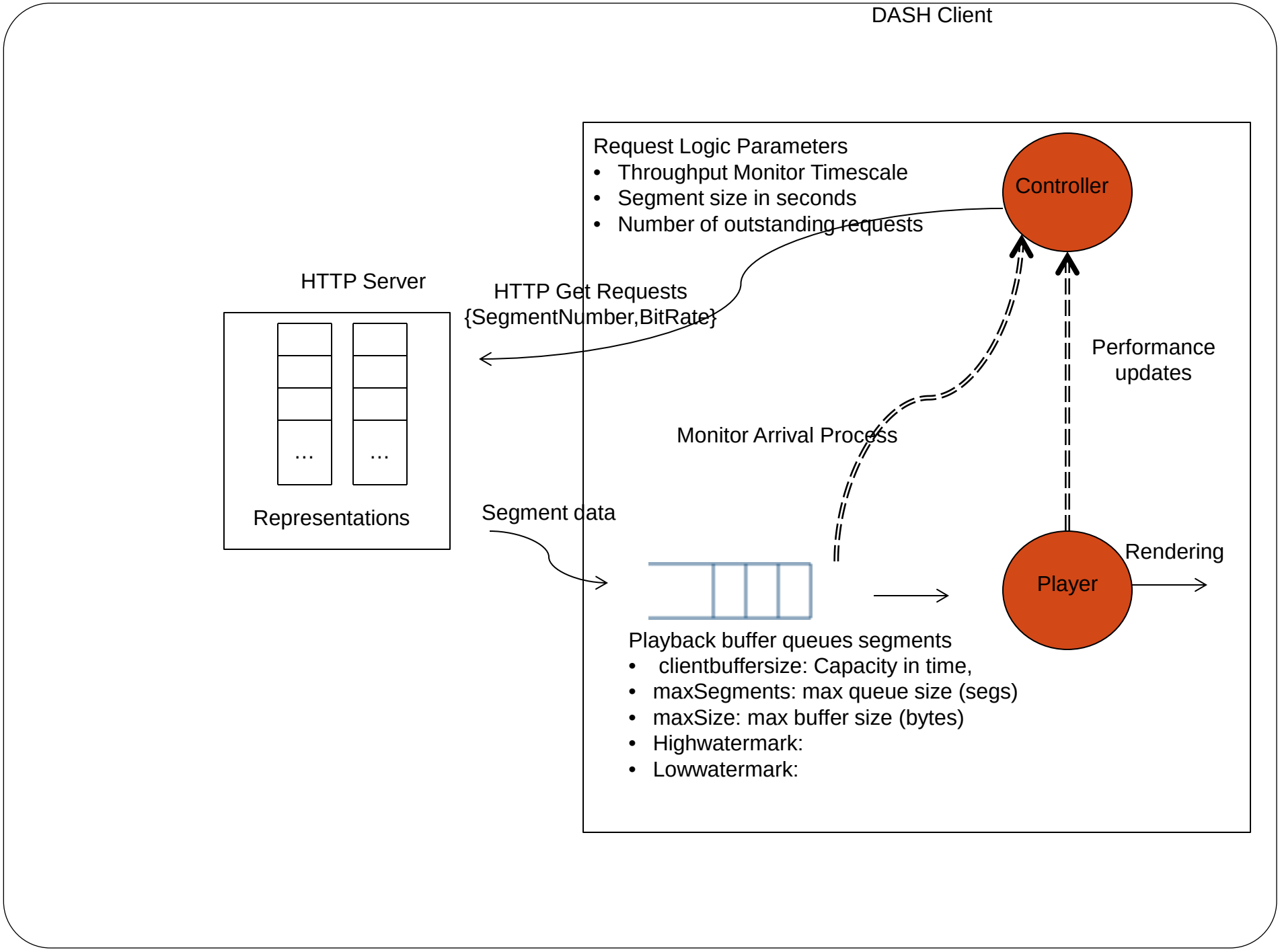
- maxCapacity: maximum queue capacity (2048 packets)
- target_delay: target max allowed queue delay (0.015 seconds)
- Tupdate: Frequency of updates to drop probability (0.016 seconds)
- max_burst: maximum allowed burst allowed (0.142 seconds)

When an acceptable channel becomes available and the queue is NOT empty

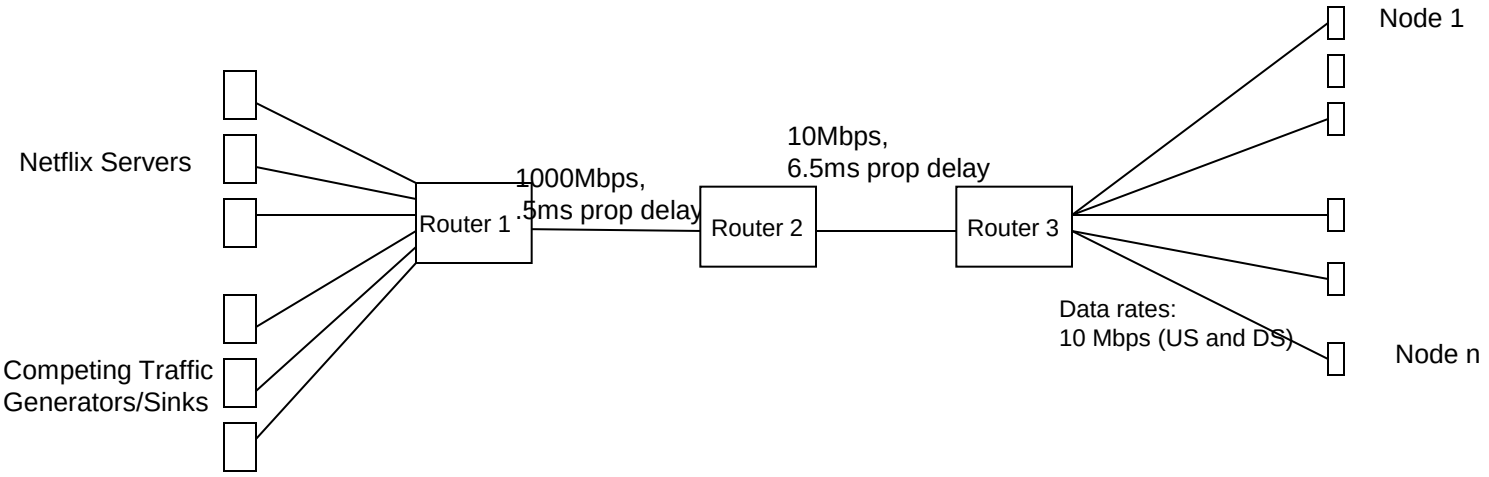
```
curTime = getCurrentTime();
pkt = dequeue();
if ( (q.getLevel() >= target_delay) && (dq_count == -1) ) {
  dq_start = curTime;
  dq_count = 0;
}
//in a measurement cycle
if (dq_count != -1) {
  dq_count ++;
  // done with a measurement cycle
  if (dq_count >= target_delay) {
    double tmp = curTime - dq_start;
     departure_rate = 0.9* departure_rate +0.1*dq_count/tmp;

    // restart a measurement cycle if there is enough data
    if (q.curSize() > target_delay) {
      dq_start = curTime;
      dq_count = 0;
    }  else {
      dq_count = -1;
    }
    // update burst allowance
    if (burst_allowance > 0)
      burst_allowance -= tmp;
  }
}
```

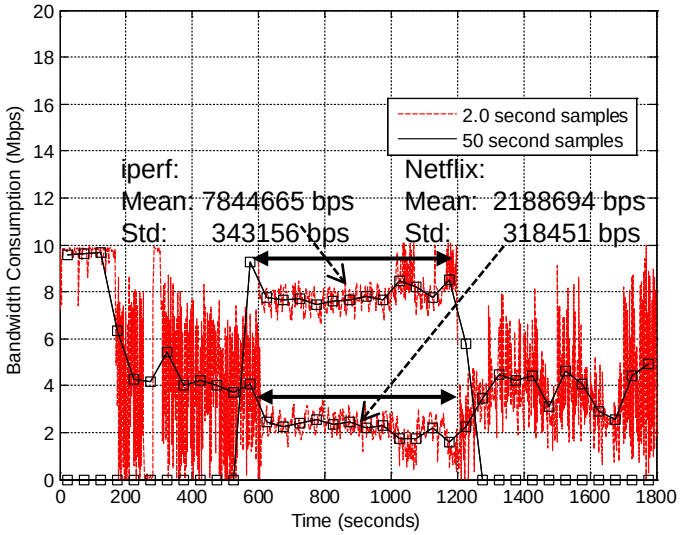
State variables:

- Pdrop : drop probability
- lastUpdateTime : last time the Pdrop was updated
- departure_rate: estimated departure rate over last measurement period (bps)
- cur_delay: The current expected packet delay
- prev_delay : the previous estimated queue delay (seconds)
- burst_allowance: current level of burst supported (init to max_burst)
- dq_count: amount of data dequed in current measurement cycle
- dq_start: start of current measurement cycle
- alpha: Pdrop computation weight for current (0.25)
- beta : Pdrop computation weight for previous (2.5)

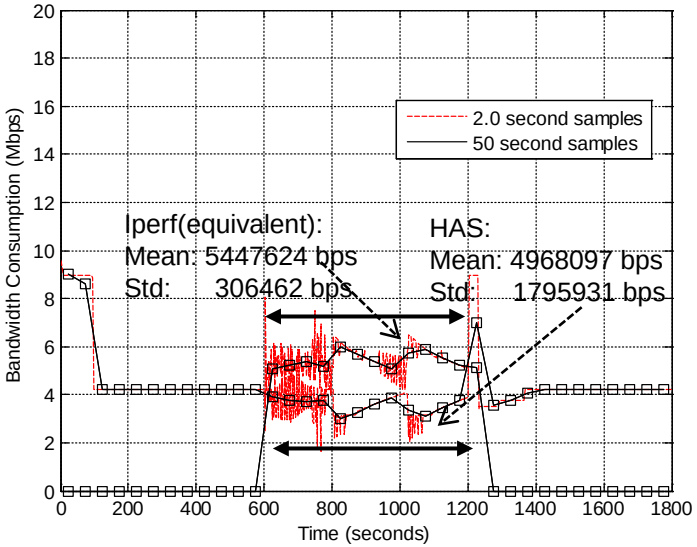

DASH Client
Request Logic Parameters
• Throughput Monitor Timescale
• Segment size in seconds
• Number of outstanding requests
Controller
HTTP Server
HTTP Get Requests
{SegmentNumber,BitRate}
Performance
updates
Monitor Arrival Process
Representations
Segment data
Rendering
Player
Playback buffer queues segments
• clientbuffersize: Capacity in time,
• maxSegments: max queue size (segs)
• maxSize: max buffer size (bytes)
• Highwatermark:
• Lowwatermark:

Network Diagram





a. Measurement Result: Competing iperf flow 600-1200 seconds



b. Simulation Result: Competing Iperf flow 600-1200 seconds

Measurement and Simulation Results (Netflix competing with iperf flow)