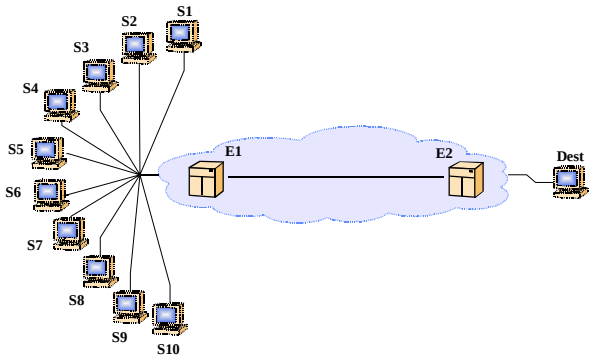
S−38.180: Quality of Service in Internet

Exercise 2: Rate Control and Queue Management

Exercise

- Comparison of different rate control methods in Internet.
  - Methods which are used are:
    - Token bucket
    - Time sliding window
- Comparison of RED and Tail Drop queue management algorithms

Network Topology



Task

- Investigate the operation of
  - Two rate control algorithm
    - Token Bucket
    - Time Sliding Window
- See how fairly rate control algortihms operate in otherwise same network environments
  - Do not overload network with CIRs
  - Use same and different CIRs for different client pairs
- What is the difference between responsive and unresponsive traffic flows
  - Overload the contract (CIR) with UDP (rate)
- What is the effect of additional free parameter (CBS) in contrast to TSW
  - Is there a CBS value which corresponds to TSW operation

Rate Control

- Strict policing is used
  - Packets which do not conform the contract are lost
    - Ns2 implements this feature by using virtual queue (lower precedence in same physical queue) with 100% packet loss

$qE1E2 configQ 0 1 0 0 1

Queue
Precedence
Min_TH
Max_TH
P_Max

- Single class of traffic
  - Traffic is mixed within the buffer together and RED is used to control the queue.

Task

- Investigate the operation of
  - Two queue management algorithms
    - Random Early Detection
    - Tail Drop
- Compare the syncronization effect with Tail Drop and RED
  - Use different parameter setting for RED to see the sensitivity (visual investigation is enough)
- Examine the "Great Equalizer" effect of RED in differentiated operation
  - Use different CIRs for connection pairs. Sum of the CIRs should be 1.1 −> 1.5 times the link capacity
  - Compare the actual transfer rate of individual connections to their CIRs

Environment

- Source population (traffic mix):
  - Population
    - 2 UDP sources
    - 8 TCP sources
  - Responsivenes
    - Unresponsive traffic (UDP)
    - Responsive traffic (TCP)
- Greedy FTP uses the TCP connection
  - It sends traffic when ever there is free transmission window
- CBR source uses the UDP
  - Traffic is sent with constant intervals

Rate control

- Implemented as policies:
  - $qE2E1 addPolicyEntry [$dest id] [$s1 id] TSW2CM 20 $cir1

Queue to manipulate
Controlled connection
Policer to be used
Initial codepoint
Contracted rate or size

Policers

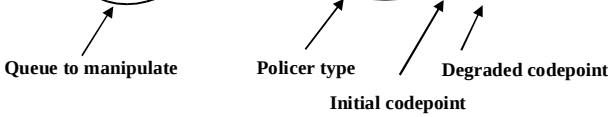
- TSW2CM:
  - Time Sliding Window Meter with Two Color Marker
  - *Queue* addPolicyEntry [*Destination* id] [*Source* id] TSW2CM *Initial Codepoint CIR*
  - Packets are marked to lower precedence probabilistically when CIR is exceeded

Policers

- Token Bucket:
  - *Queue* addPolicyEntry [*Destination* id] [*Source* id] tokenBucket *Initial Codepoint CIR CBS*
  - Packets are marked to lower precedence when bucket is empty
  - CIR = Token rate * Token size
  - CBS = Size of the token bucket

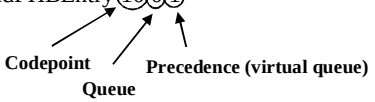
Policers

- Action of policers is defined
- $qE1E2 addPolicerEntry TSW2CM 10 11



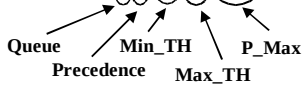
Forwarding

- Each packet with assigned codepoint need some forwarding action to be associated
- This is done by selecting queue and precedence based on the codepoint
  - $qE1E2 addPHBEntry 10 0 1



Forwarding

- Each virtual queue (precedence) is associated a RED algorithm with own parameters:
  - $qE1E2 configQ 0 1 10 40 0.10



Sources

- UDP source is defined by
  - Protocol source
    - Packet size
  - Protocol destination
    - Null
  - Application
    - Packet size
    - Sending rate

```
set udp1 [new Agent/UDP]
$ns attach-agent $s1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $packetSize
$udp1 set packetSize_ $packetSize
$cbr1 set rate_ $rate1
set null1 [new Agent/Null]
$ns attach-agent $dest $null1
$ns connect $udp1 $null1
```


Sources

- TCP source used in this exercise is special class developed for educational purposes

```
set tcp3 [new Agent/TCP/RFC793edu]
$ns attach-agent $s3 $tcp3
$tcp3 set window_ $maxwnd
$tcp3 set rtxcur_init_ $initialtimeout
$tcp3 set add793jacobsonrtt_ $jacobsonrtt
$tcp3 set add793karnrtt_ $karnrtt
$tcp3 set add793rto_ $rto
$tcp3 set add793syn_ $syn
$tcp3 set add793expbackoff_ $expbackoff
$tcp3 set add793fastrtx_ $fastrtx
$tcp3 set add793slowstart_ $slowstart
$tcp3 set add793additiveinc_ $additiveinc
$tcp3 set add793exponinc_ $exponinc

$ns at 0.001 "$tcp3 set ssthresh_ 1"

set sink3 [new Agent/TCPSink]
$ns attach-agent $dest $sink3
$ns connect $tcp3 $sink3
```


Simulator output

- Event file where all
  - Enqueue (+)
  - Dequeue (−)
  - Drop (d)
  - Receive (r)

```
r 6.6938 4 5 tcp 1000 ------- 1 0.0 2.0 657 1301
+ 6.6938 5 2 tcp 1000 ------- 1 0.0 2.0 657 1301
- 6.6938 5 2 tcp 1000 ------- 1 0.0 2.0 657 1301
r 6.694155 1 4 tcp 1000 ------- 2 1.0 3.0 10 1356
+ 6.694155 4 5 tcp 1000 ------- 2 1.0 3.0 10 1356
d 6.694155 4 5 tcp 1000 ------- 2 1.0 3.0 10 1356
r 6.6946 0 4 tcp 1000 ------- 1 0.0 2.0 684 1357
```

are presented in tabulated fashion

Simulator output

- Two files are generated:
  - Out.rec: Contains all receive events in simulation
    - Can be prosessed with perl script "Analysis.pl" or some other means

Connection pair	Bytes	Transfer time	Transfer Rate
0	1232000	50.169312	196454.757043509
1	1202000	50.169312	191670.956141476
2	1262040	50.0832	201590.952654782
3	1243040	50.02704	198778.90037068
4	1253040	50.06688	200218.587617203
5	1276040	50.348192	202754.450447794
6	1262040	49.98256	201996.856503548
7	1270040	50.252224	202186.474373751
8	1261040	50.12624	201258.263137231
9	1212040	50.363552	192526.531885599

Simulator output

- Two files are generated:
  - Out.wnd: Contains window values of TCP connections in sampling times
    - Can be plotted with gnuplot script "plot.p"