June 19, 2019

ACK SHAPER WITHOUT BUFFER

Erico Vanini

Rong Pan

Parvin Taheri

Jason Marinshaw

ACK SHAPER WITHOUT BUFFER

AUTHORS:
Erico Vanini
Rong Pan
Parvin Taheri
Jason Marinshaw

ABSTRACT

Techniques are described herein to shape Transmission Control Protocol (TCP) traffic by pacing ACK messages artificially. This may enable the sender to conform to a desired rate. TCP flow rates may thereby be effectively controlled without a large and costly packet buffer.

DETAILED DESCRIPTION

In today's networks, the ability to meter and limit network resources is becoming increasingly important. This concept can be found everywhere, from the mobile network perspective to video streaming to the cloud and data center. The infrastructure owner wants to optimize network resources by limiting or prioritizing specific types of traffic, limiting speed for specific users, etc. Typically this is achieved using a shaper which delays packets to ensure a uniform (or shaped) traffic pattern. The typical way to shape is to delay packets by buffering them and releasing them in an orderly (or shaped) pattern.

Techniques are described herein to shape a Transmission Control Protocol (TCP) flow rate without the need for a buffer. Briefly, the acknowledgement of packets (i.e., the ACK packets) is slowed, giving the sender the impression of transmitting on a lower rate link. The TCP congestion control mechanism (REQ/ACK) at the source will adapt to this condition and lower the speed of the transfers.

The typical approach to slow the ACK messages is to buffer all ACK packets belonging to the flow in question at the switch wishing to slow the traffic, and release them at a specific rate. The issue with this approach is that the switch may need to store a significant number of ACK packets, and potentially a large buffer is required.

To slow down ACK messages without buffering, the switch tracks only the last ACK number sent and the last ACK number received by that switch (and one packet of the

flow, depending on the implementation). Now the shaper only needs to shape the ACK number and avoid buffering every ACK message.

Once these two ACK numbers and a target rate for the flow are acquired, the switch can artificially generate valid ACK messages. The time between two subsequently generated ACK messages and bytes acknowledged (the difference between the ACK field value) may trigger TCP congestion control at the transmission to reduce its rate to a maximum of the target rate when in congestion avoidance mode and by two when in slow start mode. In order to avoid doubling the target rate, these techniques provide a mechanism to detect the slow start phase in order to limit the rate to the target rate. When TCP is in this phase, this detector simply monitors the real rate of incoming ACKs and compares it to the target rate. If these two rates differ by a large amount, the TCP is sending faster than desired, possibly because of the slow start phase.

The ACK messages may be generated using the following formula:

target rate = ACK bytes / period between two artificially generated ACK messages

where all three variables may be defined as desired, depending on the constraints.

If the goal is to obtain a speed of 100Mbps, any value may be chosen for the two remaining variables. For example, in order to avoid sending too many packets, the system may send just one ACK message every 10ms, deriving the ACKed bytes therefrom.

In the presence of duplicate ACKs, the sender may be informed as quickly as possible about the possibility of a drop. Hence, these ACK message may proceed without shaping. However, a subsequent ACK may be delayed, as if the duplicate ACK was slowed down, to guarantee that the average flow rate is still equal to the target rate. Even though the window is cut in half, adding artificial latency (e.g., on the order of 100ms in a datacenter) to make the Round-Trip Time (RTT) appear to the TCP to be much larger than it really is, the real RTT that is exposed to the TCP after the drop can mitigate the effect of the window cut, reducing the recovery time.

With this mechanism, TCP traffic may be slowed to specific rates, which is highly desirable in times of congestion without needing to drop packets or Explicit Congestion Notifications (ECNs). Therefore, this mechanism may be deployed anywhere. The only functionality that all switches along the path must support is ACK forwarding.

The move toward utilization of new TCP flavors has always been slow and therefore well-established TCP flavors are expected to remain in use for the foreseeable future. Moreover, this mechanism may function in combination with any other modern congestion avoidance mechanism, in particular with those using ECN marking to adapt the flow speed. This mechanism need not be turned off for such connections. In fact, it will only initiate if for any reason the flow cannot avoid causing congestion. One example is when the detected congestion is larger than what is estimated by the TCP algorithm in use, or when the TCP congestion notification (e.g., the ECN marks) are lost, for example when a single switch along the path does not forward the ECN field correctly or ignores it, which is still very common in the global network. Another example is when the source does not comply with the other mechanism in place, which could simply be due to a wrong setting or a bug in the implementation. One advantage of the techniques described herein is that the ACK mechanism is a very fundamental part of TCP, and using this feature may ensure that this mechanism will operate as expected and is resilient to any external choice/feature supported (or not) along the path. These techniques replace ACK buffering with a smarter and cheaper solution.

Empirical results show the feasibility of this approach implemented for a floodlight controller. Any SDN controller may be utilized, although in that case the operations described herein may rely entirely on the Central Processing Unit (CPU) power of the controller. A hardware implementation may therefore be the optimal way to implement this functionality directly in the switch.

Figure 1 below illustrates a basic ACK shaper pseudocode algorithm.

```
//supposing timer is fix we vary only the number of bytes we ack each timer interval
setFlowRate ( flowID,  speedLimit):
        flow = new flow(flowID)
        flow.ackIncrement(speedlimit * timer)
        flowsTable.add(flow)

incomingTcpPacket( pkt) :
        flow = flowsTable .getFlow(pkt)
        flow.lastAckRecived = pkt.ackNO
        if(flow.isduplciateAck(pkt)):
                send(pkt)
        else if (pkt.hasdata):
                pkt.AckNO = flow.lastSentAck
                send(pkt)
        else :
                drop(ack)

timerTriggerAckScheduler():
        foreach flow in flowsTable :
                newAckNO= flow.lastSentAck + flow.ackIncrement
                if (newAclNO <= flow.lastAckReceived) :
                        newAckMsg = flow.createAck( newAckNO )
                        send (newAckMsg )
                        flow.lastsentAck=newAckNO
```

Figure 1

Figure 2 below illustrates throughput over time in a first example.

Omnet, 1ms avg thput, 1 flow 10G limit, 1 flow 5G limit, same destination

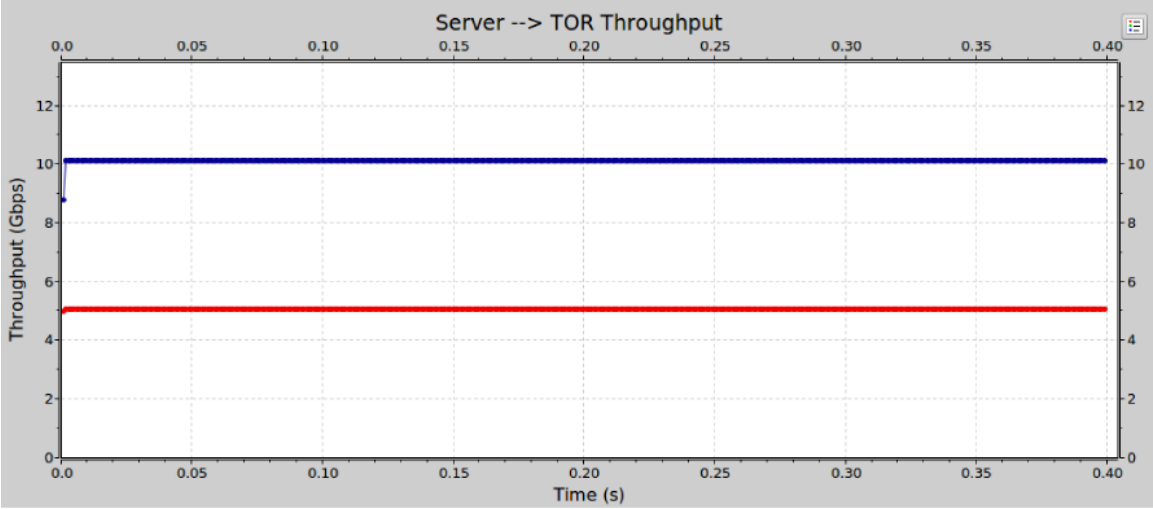

Figure 2

Figure 3 below illustrates throughput over time in a second example.

Omnet, 1ms avg thput, 1 flow 10G limit, 1 flow 5G limit, same destination link with 0.01% drops

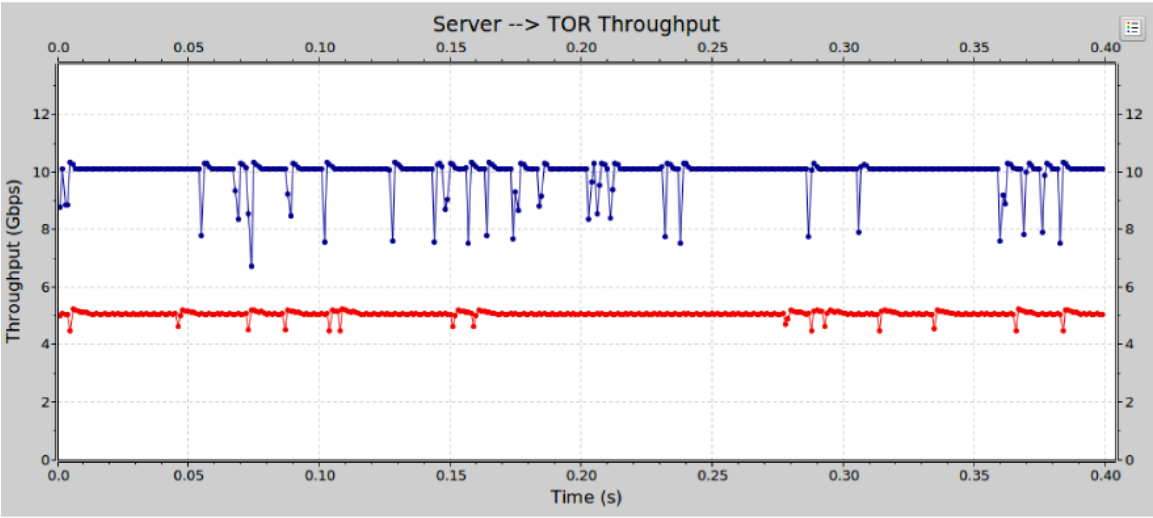

Figure 3

Figure 4 below illustrates throughput over time in a third example.

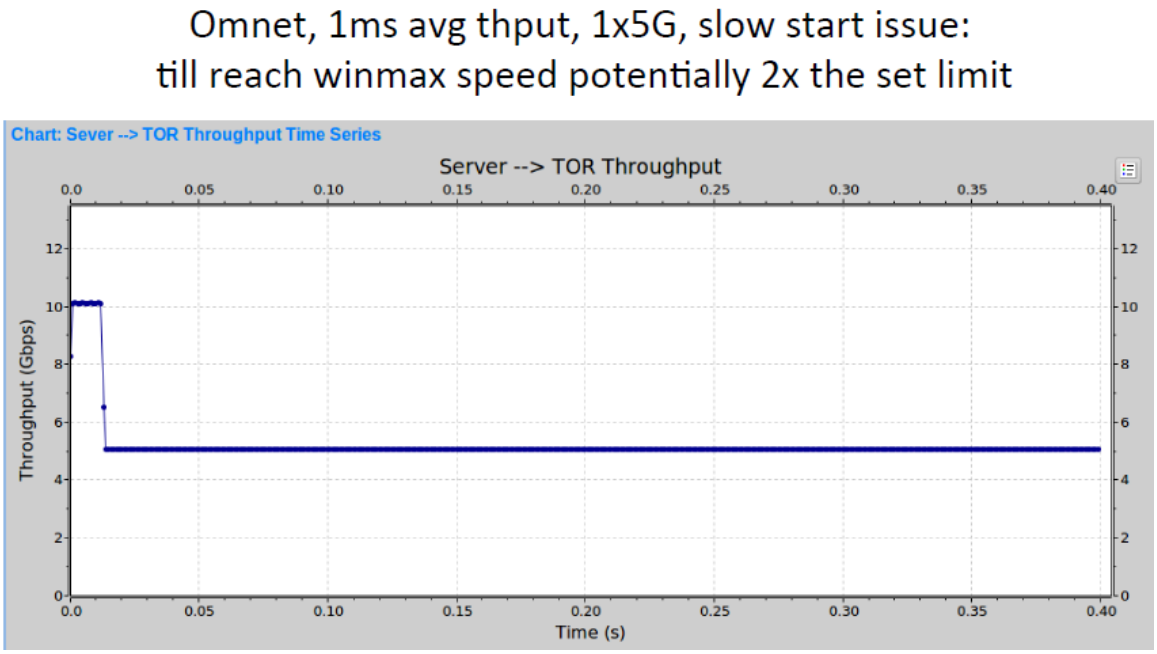

Figure 4

Figure 5 below illustrates ACK shaper pseudocode with slow start detection.

```
Change in  timerTriggerAckScheduler function:

timerTriggerAckScheduler():
        foreach flow in flowsTable :
                flow.MovingAvgBytesSent = ( 0.7 * flow.MovingAvgBytesSent )  + ( 0.3 * flow.lastAckReceived - flow.lastsentAck)
                if (flow.MovingAvgBytesSent / flow.ackIncrement > 2):  //we are in slowstart
                        flow.isInSlowstartmode = true
                else if (flow.MovingAvgBytesSent / flow.ackIncrement < 0.5):  //slowstart is finish
                        flow.isInSlowstartmode = false
                if (flow.isInSlowstartmode ):
                        newAckNO= flow.lastsentAck + ( flow.ackIncrement / 2 )
                else:
                        newAckNO= flow.lastsentAck +  flow.ackIncrement

                if (newAcINO <= flow.lastAckReceived) :
                        newAckMsg = flow.createAck( newAckNO )
                        send (newAckMsg )
                        flow.lastsentAck=newAckNO
```

Figure 5

Figure 6 below illustrates throughput over time in a fourth example. Because the time in this example is less than two microseconds, less than one packet per ACK is released. As a result, there is not a new incoming ACK for each timer trigger. Instead, a moving average is employed.

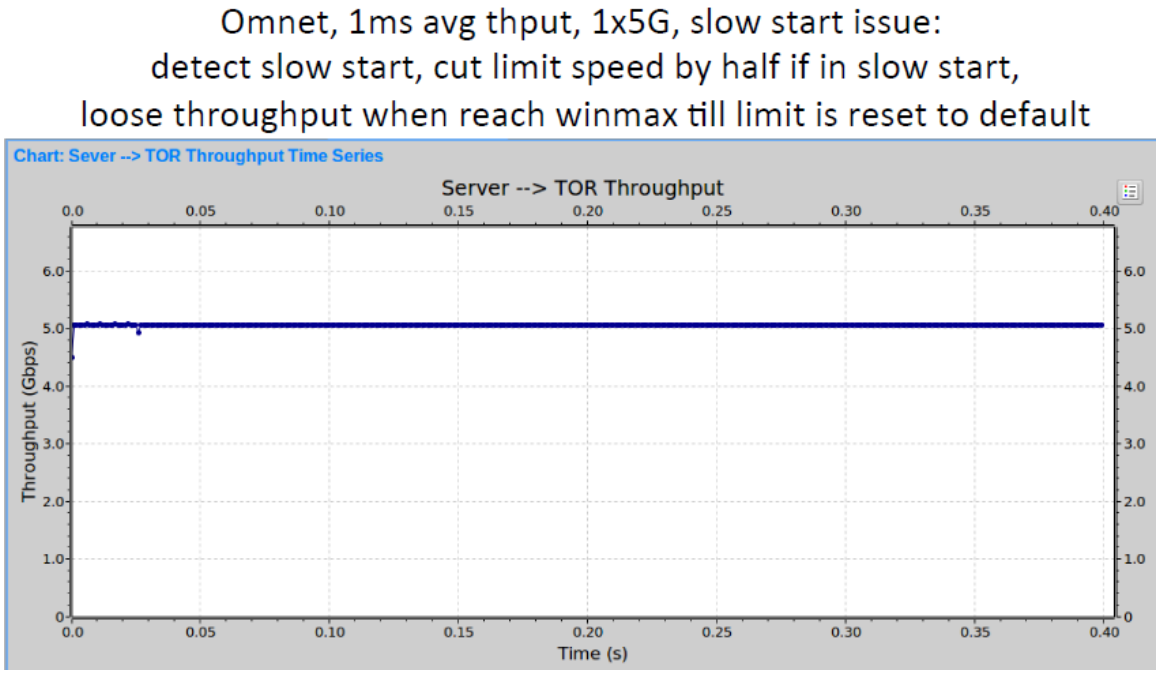

Figure 6

Figure 7 below illustrates throughput over time in a fifth example.

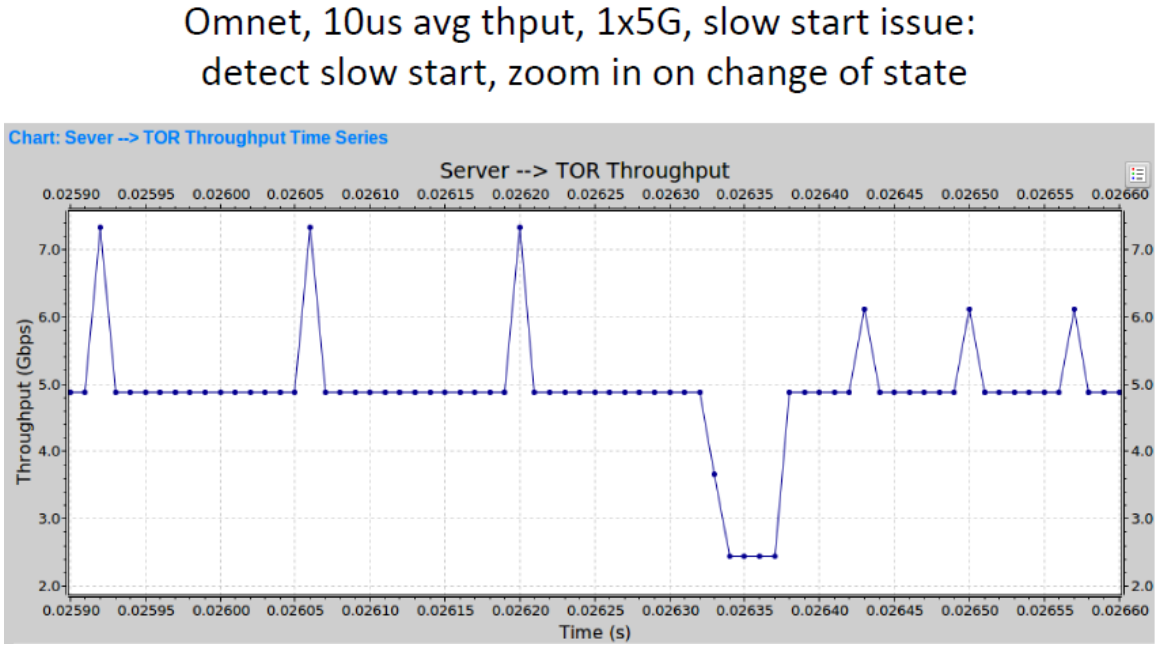

Figure 7

Figures 8-12 below illustrate results for various flows.

Mininet iperf results, 1x10M, 1x100M, 400ms RTT: without slow start detec

10M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	15.0 MBytes	12.6 Mbits/sec
[15] 10.0-20.0 sec	19.5 MBytes	16.4 Mbits/sec
[15] 20.0-30.0 sec	12.2 MBytes	10.3 Mbits/sec
[15] 30.0-40.0 sec	12.2 MBytes	10.3 Mbits/sec
[15] 40.0-50.0 sec	12.2 MBytes	10.3 Mbits/sec
[15] 50.0-60.0 sec	12.4 MBytes	10.4 Mbits/sec
[15] 60.0-70.0 sec	8.12 MBytes	6.82 Mbits/sec
[15] 70.0-80.0 sec	12.2 MBytes	10.3 Mbits/sec
[15] 80.0-90.0 sec	12.4 MBytes	10.4 Mbits/sec
[15] 90.0-100.0 sec	12.2 MBytes	10.3 Mbits/sec
[15] 0.0-101.0 sec	**129 MBytes**	**10.7 Mbits/sec**

100M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	134 MBytes	112 Mbits/sec
[15] 10.0-20.0 sec	120 MBytes	100 Mbits/sec
[15] 20.0-30.0 sec	119 MBytes	100 Mbits/sec
[15] 30.0-40.0 sec	116 MBytes	97.1 Mbits/sec
[15] 40.0-50.0 sec	118 MBytes	99.1 Mbits/sec
[15] 50.0-60.0 sec	119 MBytes	100 Mbits/sec
[15] 60.0-70.0 sec	119 MBytes	100 Mbits/sec
[15] 70.0-80.0 sec	119 MBytes	100 Mbits/sec
[15] 80.0-90.0 sec	119 MBytes	100 Mbits/sec
[15] 90.0-100.0 sec	119 MBytes	100 Mbits/sec
[15] 0.0-100.1 sec	**1.18 GBytes**	**101 Mbits/sec**

Figure 8

Mininet iperf results, 1x20M, 1x100M, 400ms RTT:

20M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	29.2 MBytes	24.5 Mbits/sec
[15] 10.0-20.0 sec	29.1 MBytes	24.4 Mbits/sec
[15] 20.0-30.0 sec	24.5 MBytes	20.6 Mbits/sec
[15] 30.0-40.0 sec	24.6 MBytes	20.7 Mbits/sec
[15] 40.0-50.0 sec	20.5 MBytes	17.2 Mbits/sec
[15] 50.0-60.0 sec	24.5 MBytes	20.6 Mbits/sec
[15] 60.0-70.0 sec	24.6 MBytes	20.7 Mbits/sec
[15] 70.0-80.0 sec	24.6 MBytes	20.7 Mbits/sec
[15] 80.0-90.0 sec	24.5 MBytes	20.6 Mbits/sec
[15] 90.0-100.0 sec	20.5 MBytes	17.2 Mbits/sec
[15] 0.0-100.0 sec	**247 MBytes**	**20.7 Mbits/sec**

100M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	129 MBytes	108 Mbits/sec
[15] 10.0-20.0 sec	120 MBytes	100 Mbits/sec
[15] 20.0-30.0 sec	119 MBytes	100 Mbits/sec
[15] 30.0-40.0 sec	120 MBytes	100 Mbits/sec
[15] 40.0-50.0 sec	119 MBytes	100 Mbits/sec
[15] 50.0-60.0 sec	119 MBytes	100 Mbits/sec
[15] 60.0-70.0 sec	119 MBytes	100 Mbits/sec
[15] 70.0-80.0 sec	119 MBytes	100 Mbits/sec
[15] 80.0-90.0 sec	120 MBytes	100 Mbits/sec
[15] 90.0-100.0 sec	119 MBytes	100 Mbits/sec
[15] 0.0-100.2 sec	**1.18 GBytes**	**101 Mbits/sec**

Figure 9

Mininet iperf results, 1x50M, 1x100M, 400ms RTT:

50M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	68.6 MBytes	57.6 Mbits/sec
[15] 10.0-20.0 sec	64.1 MBytes	53.8 Mbits/sec
[15] 20.0-30.0 sec	55.8 MBytes	46.8 Mbits/sec
[15] 30.0-40.0 sec	60.2 MBytes	50.5 Mbits/sec
[15] 40.0-50.0 sec	60.2 MBytes	50.5 Mbits/sec
[15] 50.0-60.0 sec	60.2 MBytes	50.5 Mbits/sec
[15] 60.0-70.0 sec	60.4 MBytes	50.6 Mbits/sec
[15] 70.0-80.0 sec	60.2 MBytes	50.5 Mbits/sec
[15] 80.0-90.0 sec	60.2 MBytes	50.5 Mbits/sec
[15] 90.0-100.0 sec	55.6 MBytes	46.7 Mbits/sec
[15] 0.0-100.1 sec	**606 MBytes**	**50.8 Mbits/sec**

100M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	131 MBytes	110 Mbits/sec
[15] 10.0-20.0 sec	119 MBytes	100 Mbits/sec
[15] 20.0-30.0 sec	119 MBytes	100 Mbits/sec
[15] 30.0-40.0 sec	120 MBytes	100 Mbits/sec
[15] 40.0-50.0 sec	119 MBytes	100 Mbits/sec
[15] 50.0-60.0 sec	119 MBytes	100 Mbits/sec
[15] 60.0-70.0 sec	119 MBytes	100 Mbits/sec
[15] 70.0-80.0 sec	119 MBytes	100 Mbits/sec
[15] 80.0-90.0 sec	119 MBytes	100 Mbits/sec
[15] 90.0-100.0 sec	120 MBytes	100 Mbits/sec
[15] 0.0-100.4 sec	**1.18 GBytes**	**101 Mbits/sec**

Figure 10

Mininet iperf results, 1x50M, 1x100M, 400msRTT, ack timer 10ms:

10M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	12.1 MBytes	10.2 Mbits/sec
[15] 10.0-20.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 20.0-30.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 30.0-40.0 sec	12.0 MBytes	10.1 Mbits/sec
[15] 40.0-50.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 50.0-60.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 60.0-70.0 sec	12.0 MBytes	10.1 Mbits/sec
[15] 70.0-80.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 80.0-90.0 sec	11.9 MBytes	9.96 Mbits/sec
[15] 90.0-100.0 sec	12.0 MBytes	10.1 Mbits/sec
[15] 0.0-100.2 sec	**120 MBytes**	**10.0 Mbits/sec**

100M flow

[ID] Interval	Transfer	Bandwidth
[15] 0.0-10.0 sec	119 MBytes	99.5 Mbits/sec
[15] 10.0-20.0 sec	118 MBytes	99.3 Mbits/sec
[15] 20.0-30.0 sec	119 MBytes	99.8 Mbits/sec
[15] 30.0-40.0 sec	119 MBytes	99.5 Mbits/sec
[15] 40.0-50.0 sec	118 MBytes	99.4 Mbits/sec
[15] 50.0-60.0 sec	119 MBytes	99.5 Mbits/sec
[15] 60.0-70.0 sec	119 MBytes	99.7 Mbits/sec
[15] 70.0-80.0 sec	119 MBytes	99.9 Mbits/sec
[15] 80.0-90.0 sec	119 MBytes	99.7 Mbits/sec
[15] 90.0-100.0 sec	118 MBytes	99.3 Mbits/sec
[15] 0.0-100.0 sec	**1.16 GBytes**	**99.6 Mbits/sec**

Figure 11

Mininet iperf results, 8x5M, 8x10M, 4msRTT, ack timer 10ms:

10M flow

[ID] Interval	Transfer	Bandwidth
[43] 0.0- 2.0 sec	2.62 MBytes	11.0 Mbits/sec
[43] 2.0- 4.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 4.0- 6.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 6.0- 8.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 8.0-10.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 10.0-12.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 12.0-14.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 14.0-16.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 16.0-18.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 18.0-20.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 20.0-22.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 22.0-24.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 24.0-26.0 sec	2.25 MBytes	9.44 Mbits/sec
[43] 26.0-28.0 sec	2.25 MBytes	9.44 Mbits/sec
[43] 28.0-30.0 sec	2.00 MBytes	8.39 Mbits/sec
[43] 30.0-32.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 32.0-34.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 34.0-36.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 36.0-38.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 38.0-40.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 40.0-42.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 42.0-44.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 44.0-46.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 46.0-48.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 48.0-50.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 50.0-52.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 52.0-54.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 54.0-56.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 56.0-58.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 58.0-60.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 60.0-62.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 62.0-64.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 64.0-66.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 66.0-68.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 68.0-70.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 70.0-72.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 72.0-74.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 74.0-76.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 76.0-78.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 78.0-80.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 80.0-82.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 82.0-84.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 84.0-86.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 86.0-88.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 88.0-90.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 90.0-92.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 92.0-94.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 94.0-96.0 sec	2.38 MBytes	9.96 Mbits/sec
[43] 96.0-98.0 sec	2.50 MBytes	10.5 Mbits/sec
[43] 98.0-100.0 sec	2.38 MBytes	9.96 Mbits/sec

5M flow

[ID] Interval	Transfer	Bandwidth
[43] 0.0- 2.0 sec	1.38 MBytes	5.77 Mbits/sec
[43] 2.0- 4.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 4.0- 6.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 6.0- 8.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 8.0-10.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 10.0-12.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 12.0-14.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 14.0-16.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 16.0-18.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 18.0-20.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 20.0-22.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 22.0-24.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 24.0-26.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 26.0-28.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 28.0-30.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 30.0-32.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 32.0-34.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 34.0-36.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 36.0-38.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 38.0-40.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 40.0-42.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 42.0-44.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 44.0-46.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 46.0-48.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 48.0-50.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 50.0-52.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 52.0-54.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 54.0-56.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 56.0-58.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 58.0-60.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 60.0-62.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 62.0-64.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 64.0-66.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 66.0-68.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 68.0-70.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 70.0-72.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 72.0-74.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 74.0-76.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 76.0-78.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 78.0-80.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 80.0-82.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 82.0-84.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 84.0-86.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 86.0-88.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 88.0-90.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 90.0-92.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 92.0-94.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 94.0-96.0 sec	1.25 MBytes	5.24 Mbits/sec
[43] 96.0-98.0 sec	1.12 MBytes	4.72 Mbits/sec
[43] 98.0-100.0 sec	1.25 MBytes	5.24 Mbits/sec

16 flows overall avg

[ID] Interval	Transfer	Bandwidth
[44] 0.0-100.3 sec	59.9 MBytes	5.01 Mbits/sec
[45] 0.0-100.2 sec	121 MBytes	10.1 Mbits/sec
[46] 0.0-100.4 sec	59.9 MBytes	5.01 Mbits/sec
[47] 0.0-100.2 sec	121 MBytes	10.1 Mbits/sec
[48] 0.0-100.3 sec	59.9 MBytes	5.01 Mbits/sec
[49] 0.0-100.1 sec	121 MBytes	10.1 Mbits/sec
[50] 0.0-100.3 sec	59.9 MBytes	5.01 Mbits/sec
[51] 0.0-100.1 sec	121 MBytes	10.1 Mbits/sec
[52] 0.0-100.4 sec	59.9 MBytes	5.00 Mbits/sec
[53] 0.0-100.1 sec	121 MBytes	10.1 Mbits/sec
[54] 0.0-100.3 sec	59.9 MBytes	5.01 Mbits/sec
[55] 0.0-100.2 sec	121 MBytes	10.1 Mbits/sec
[56] 0.0-100.4 sec	59.9 MBytes	5.01 Mbits/sec
[57] 0.0-100.1 sec	121 MBytes	10.1 Mbits/sec
[58] 0.0-100.4 sec	59.9 MBytes	5.00 Mbits/sec
[59] 0.0-100.1 sec	121 MBytes	10.1 Mbits/sec

Figure 12

These techniques may be implemented as a policer or shaper, or as an in-switch reaction mechanism in combination with early congestion detection to avoid congestion and buffer buildup and ultimately drops. This improves on current approaches by reducing the overhead/cost of the implementation.

In summary, techniques are described herein to shape TCP traffic by pacing ACK messages artificially. This may enable the sender to conform to a desired rate. TCP flow rates may thereby be effectively controlled without a large and costly packet buffer.