RIGHT-SIZING CABLE MODEM BUFFERS FOR MAXIMUM APPLICATION PERFORMANCE

Greg White (CableLabs®)
Richard Woundy, Yiu Lee, Carl Williams (Comcast)

Abstract

The sizing of the cable modem (CM) upstream buffer can significantly impact application performance and end-user quality of experience. Too little buffering will result in poor TCP throughput in the upstream direction. Too much buffering will have a negative impact on latency sensitive applications when the upstream link is heavily utilized. Studies have indicated that cable modems in the field today have perhaps an order of magnitude greater buffering than is needed to ensure TCP throughput, with the result being a sluggish user experience for web browsing, and outright failure of VoIP and real-time games at times when the upstream is being used simultaneously by TCP.

This paper provides a background on the network dynamics involved, describes a new feature in the DOCSIS® 3.0 specification that allows the operator to tune the upstream buffer, and provides some initial experimental work that seeks to provide guidance to operators in the use of this feature.

INTRODUCTION

Buffering of data traffic is a critical function implemented by network elements (e.g. hosts, switches, routers) that serves to minimize packet loss and maximize efficient use of the next-hop link. It is generally thought that for the Transmission Control Protocol (TCP) to work most efficiently, each element in a network needs to support sufficient buffering to allow the TCP window to open up to a value greater than or equal to the product of the next hop link bandwidth and the total round trip time for the TCP session (the bandwidth delay product). In many residential broadband scenarios, the cable modem is the head of the bottleneck link in the upstream direction, and as a result its buffering implementation can have a significant impact on performance – either imposing limits on upstream data rates or increasing delay for latency sensitive applications.

Historically, upstream buffer implementations in CMs have been sized (statically) by the modem vendor, and the network operator has had no control or visibility as to the size of the buffer implementation. In order to ensure optimal performance across the wide range of potential upstream configured data rates, modem suppliers have sized their upstream buffers such that there is sufficient buffering for TCP to work well at the highest possible upstream data rate, and at the greatest expected round trip time (i.e. the greatest possible bandwidth delay product). In the majority of deployment scenarios, the configured upstream data rate is significantly less than the highest possible rate, and the average TCP round trip time is significantly less than the greatest expected. The result is that in the majority of cases the cable modem has an upstream buffer that greatly exceeds what is necessary to ensure optimal TCP performance.

In today's mix of upstream TCP and UDP traffic, some of which is latency sensitive, some of which is not, the impact of a vastly oversized upstream buffer is that the upstream TCP sessions attempt to keep the buffer as full as possible, and the latency sensitive traffic can suffer enormously. The result is poorer than expected application performance for VoIP, gaming, web surfing, and other applications. Studies have shown upstream buffering in deployed cable modems on the order of multiple seconds.

Recent work has established a standardized means by which a cable operator can set the upstream buffer size at the cable modem via the modem's configuration file (alongside other service defining parameters, such as the maximum traffic rate). This paper provides an introduction to this new capability, investigates the performance impact of buffer size on a range of traffic scenarios, and seeks to provide some initial guidance for operators to properly configure buffer size for their service offerings.

RESEARCH AND BEST PRACTICES RELATING TO BUFFER SIZING

The TCP [2] is a connection-oriented, end-to-end protocol that enables the reliable transmission of data across the Internet. The TCP is designed to recover data that is damaged, lost, duplicated, or delivered out of order by the network.

The original TCP specification provides reliability and end-to-end flow control through the use of sequence numbers and the use of a "receive window". All transmitted data is tracked by sequence number, which enables the TCP receiver to detect missing or duplicate data. The receive window enables flow control by identifying the range of sequence numbers that the receiver is prepared to receive, to prevent overflow of the receiver's data buffer space.

However, these controls were insufficient to avoid "congestion collapse" in the Internet in the mid-1980s [4]. Van Jacobson [1] developed the initial mechanisms for TCP congestion control, which many researchers have continued to extend and to refine [6]. In particular, TCP congestion control utilizes a "congestion window" to limit the transmission rate of the sender in response to network congestion indications.

Two key congestion control algorithms are "slow start" and "congestion avoidance". In slow start, the sender increases the congestion window by one segment for each TCP acknowledgement (ACK) received; this action doubles the transmission rate for each roundtrip time, leading to exponential growth. Once in congestion avoidance, the sender increases the congestion window by one segment per roundtrip time; this action increases the transmission rate linearly. If a segment is dropped due to a timeout, the threshold for entering congestion avoidance is set to one-half the current congestion window, and the congestion window itself is reset to one segment. This "sawtooth" behavior is illustrated in Figure 1 [16].

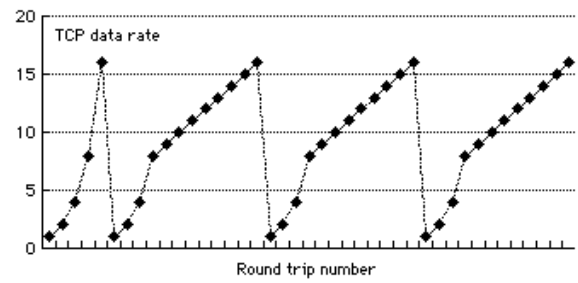

Figure 1

Modern TCP implementations also implement "fast retransmit" and "fast recovery" algorithms. According to fast retransmit, when the sender receives three duplicate ACKs, the sender assumes a segment has been lost and re-transmits that segment. If that segment is subsequently transmitted successfully, then under fast recovery the sender cuts its congestion window in half (rather than resetting to one segment).

Using modern congestion control, the TCP sender is constantly attempting to increase its transmission rate to the maximum possible, and it decreases the transmission rate in the presence of segment loss. As a result of this behavior, the TCP sender will send packets at a continuously increasing rate, filling the

buffer of the network device at the head of the bottleneck link, until one or more packets are dropped. (This discussion ignores the presence of Active Queue Management [3] or Explicit Congestion Notification [5], neither of which are commonly implemented in broadband modems.)

Since the TCP sender's behavior is to keep the buffer full at the bottleneck link, it is helpful to review the research recommendations for the size of network buffers.

Back in 1994, Villamizar and Song [7] provided performance test results using up to eight TCP flows over a national backbone infrastructure. Their results led to the rule-of-thumb that bottleneck links need a buffer size equal to the bandwidth delay product (BDP) – that is, equal to the available bandwidth at the bottleneck multiplied by the roundtrip time of the TCP sessions – for the TCP flows to saturate the bottleneck link. A decade later, Appenzeller, Keslassy and McKeown [8] confirmed the BDP rule-of-thumb for a single long-lived TCP flow, which may apply best to a bottleneck link at the edge of the network, such as a broadband modem's upstream link.

Since that time, the topic of buffering at network hops has become an active one, with some beginning to look at the impact that the rule-of-thumb has on non-TCP traffic, as well as whether the rule-of-thumb holds up under more real-world conditions. Vishwanath, et al. [10] provide a good survey of recent work in this area.

If the optimal buffer size for TCP performance is presumed to be equal to the bandwidth delay product, then one might inquire how that compares to the buffer size implemented in current broadband modems. In 2007, Dischinger et al. [9] measured the performance of a number of DSL and cable operators in Europe and in North America. While the measured broadband modem minimum delay was typically under 15 milliseconds, the upstream buffer sizes were found to be quite large – resulting in 600 milliseconds of queuing delay for most DSL links, and several seconds for many cable modems. These results suggest that broadband buffer sizes are much larger than the BDP rule-of-thumb.

Gettys [17] shows that such large buffer sizes, or "bufferbloat", is an issue today with broadband modems both for wireline and for wireless networks. As one example, some have observed six seconds of latency in 3G wireless networks related to queuing. The issue of bufferbloat has also been observed in other consumer equipment, such as home routers and laptops. Gettys points to excessive buffering at the head of the bottleneck link as being disruptive to many user applications.

CABLE MODEM UPSTREAM BUFFERING AND IMPACTS ON APPLICATION PERFORMANCE/USER EXPERIENCE

In the upstream traffic direction, the cable modem is generally the entry point of the bottleneck link, both topologically and in terms of link capacity. As a result of this, its buffer size can have a significant impact on application performance. As noted previously, TCP upload performance can be sensitive to buffer size on the bottleneck link, and furthermore will seek to keep the bottleneck link buffer full. When latency-sensitive or loss-sensitive traffic is mixed with TCP traffic in the same cable modem service flow, issues can arise as a result of the modem buffer being kept full by the TCP sessions.

One impact will be that applications will see an upstream latency that can be predicted by the buffer size divided by the upstream configured rate. For example, in the case of a 256KiB buffer and a configured upstream rate of 2Mb/s, the upstream latency would be

expected to be on the order of 1 second. More buffering will result in a proportional increase in latency.

Another impact will be that upstream applications will experience packet loss due to buffer overflow. Bulk TCP sessions are immune to packet loss resulting from buffer overflow, and in fact their congestion avoidance algorithm relies on it. However, other applications may be more sensitive. In contrast to the latency impact, the packet loss rate will increase if the buffer is undersized.

How Much Buffering Should There Be?

Commonly used VoIP and video conferencing applications such as Vonage, Skype, iChat, FaceTime, and umi, provide a better user experience when end-to-end latency is kept low. The ITU has developed models and recommendations for end-to-end delay for voice and other interactive services. The ITU-T Recommendation G.114 [11] suggests one-way latency be kept below 150ms in order to achieve "essentially transparent interactivity", and that delays greater than 400ms are unacceptable for interactive applications. Furthermore it has defined the "E-model" (ITU-T G.107 [12], G.108 [13] & G.109 [14]) used for predicting digital voice quality based on system parameters. The E-model has been reduced to a simpler form for VoIP applications by Cole & Rosenbluth, which points to significant degradation in voice quality beginning when the one-way latency exceeds ~177ms. Packet loss also degrades quality, with losses of less than 2.5% being necessary to achieve "Best" quality. Included in the packet loss statistic are any packets that exceed the ability of the receiver de-jitter buffer to deliver them isochronously.

Multiplayer online games can also be sensitive to latency and packet loss. Fast-paced, interactive games are typically the most sensitive to latency, where some games require that players have less than 130ms round-trip time between their host and the game server in order to be allowed to play, and lower is generally better.

Even more latency sensitive are games that are run on a cloud server, with a video stream sent to the player, and a control stream sent from the player back to the server. Services such as these may require a maximum of 20-50ms round-trip time across the cable operator's network.

While not typically considered to be a latency sensitive application, web-browsing activities can also be impacted by upstream latency. Consider that a webpage load consists of a cascade of perhaps a dozen or more round trips to do DNS lookups and HTTP object fetches. An upstream latency on the order of 500ms or more is likely to cause the web browsing experience to be much slower than a user might find acceptable, particularly if they are paying for a downstream connection speed measured in tens of megabits per second.

In addition, any downstream-centric TCP application (such as web browsing) has the potential to be throttled due to the latency experienced by the upstream TCP-ACK stream from the receiver. In practice this is not an issue, because many cable modems support proprietary means to accelerate TCP-ACKs by queuing them separately from bulk upstream data.

How Much Queuing Latency Is There?

Historically, cable modems have implemented static buffer sizes regardless of upstream data rate. Evidence suggests that cable modems in the field today may have buffers that are sized (using the BDP rule-of-thumb) for the maximum possible upstream data rate (~25Mb/s for DOCSIS 2.0 CMs and

~100Mb/s for current DOCSIS 3.0 CMs) and the maximum coast-to-coast Round Trip Time (RTT) that might be experienced (100ms or more).

Our observations indicate that most conventional cable modems are built with buffer size between 60KiB and 300KiB.

Since the majority of modems (particularly those in residential service) are currently operated with upstream rates in the range of 1 to 2Mb/s, the result (as corroborated by Dischinger [9]) is that the modems have buffering latencies on the order of several seconds, which may be 1 or 2 orders of magnitude greater than would be ideal.

DOCSIS 3.0 BUFFER CONTROL FUNCTIONALITY

A recent addition to the DOCSIS 3.0 specifications provides, for the first time, the ability for cable operators to tune thetransmit buffers for cable modems and CMTSs in their networks. This new feature, support for which became mandatory for cable modems submitted for CableLabs certification in early April 2011 (and is optional for CMTSs), gives the operator control of the configured buffer size for each DOCSIS Service Flow. The feature controls upstream buffering in the cable modem, and downstream buffering in the CMTS.

The new feature is referred to as Buffer Control, and is defined as a Quality of Service (QoS) Parameter of a DOCSIS Service Flow. Specifically, the Buffer Control parameter is a set of Type-Length-Value (TLV) tuples that define a range of acceptable buffer sizes for the service flow, as well as a target size. This allows the CM/CMTS implementer some flexibility on the resolution of its buffer configuration. For example, a CM implementer might choose to manage buffering in blocks of 1024 bytes, if that offered some implementation advantages. When presented with a Buffer Control TLV, such a CM would then choose a number of blocks that results in a buffer size that is within the range of acceptable sizes, and is as close to the target buffer size as possible.

The Buffer Control TLV is comprised of three parameters:

Minimum Buffer - defines the lower limit of acceptable buffer sizes. If the device cannot provide at least this amount of buffering, it will reject the service flow. If this parameter is omitted, there is no minimum buffer size.

Target Buffer - defines the desired size of the buffer. The device will select a buffer size that is as close to this value as possible, given its implementation. If this parameter is omitted, the device selects any buffer size within the allowed range, via a supplier-specific algorithm.

Maximum Buffer - defines the upper limit of acceptable buffer sizes. If the device cannot provide a buffer size that is less than or equal to this size, it will reject the service flow. If this parameter is omitted, there is no maximum buffer size.

Each of these parameters is defined in bytes, with an allowed range of 0 - 4,294,967,295 (4 GiB).

As noted, each of the three parameters is optional. If all three parameters are omitted, the interpretation by the device is that there is no limitation (minimum or maximum) on allowed buffer size for this service flow, and that the device should select a buffer size via a vendor specific algorithm. This is exactly the situation that existed prior to the introduction of this feature. As a result, if the operator does not change its modem configurations to include Buffer Control, the operator should see no difference in behavior between a

modem that supports this new feature and one that does not.

In many configuration cases it will be most appropriate to omit the Minimum Buffer and Maximum Buffer limits, and to simply set the Target Buffer. The result is that the modem will not reject the service flow due to buffering configuration, and will provide a buffer as close as the implementation allows to the Target Buffer value.

In certain cases however, the operator may wish to be assured that the buffer is within certain bounds, and so would prefer an explicit signal (i.e., a rejection of the configuration) if the modem cannot provide a buffer within those bounds. Hard limits are provided for these cases.

The cable modem is required to support buffer configurations of up to 24KiB per service flow, but is expected to support significantly more, particularly when a small number of service flows is in use. Put another way, if the Minimum Buffer parameter is set to a value less than or equal to 24576, the cable modem is guaranteed not to reject the configuration due to that parameter value. If the Minimum Buffer parameter is set to a value that is greater than 24576, then there is some risk that a modem implementation will reject the configuration.

Since they are part of the QoS Parameter Set, the Buffer Control TLVs can be set directly in the cable modem's configuration boot file; they can be set indirectly via a named Service Class defined at the CMTS; and they can be set and/or modified via the PacketCable™ Multimedia (PCMM) interface to the CMTS.

In order to use this new feature to control upstream buffering in DOCSIS 3.0 cable modems, it is necessary that the CM software be updated to a version that supports it. Furthermore, CMTS software updates are necessary in order to ensure that the CMTS properly sends the TLVs to the CM, regardless of which of the above methods is utilized.

EXPERIMENTAL WORK

Methodology

The new Buffer Control parameters described in a previous section enable us to provision the buffer size in the cable modem, and thus provide the opportunity to investigate the application performance impact in various scenarios. For the purposes of this paper, we ran three sets of tests:

In the first set of tests, we correlate the buffer size and upstream speed. The purpose of the test is to understand the impact buffer size would have on the ability of the customer to utilize his/her provisioned upstream bandwidth. As an illustration, if the round-trip time for a TCP session is 40ms and the upstream service is provisioned for 5Mb/s, the rule-of-thumb for buffer size would indicate 5000Kb/s * 0.04s = 200Kb or 24.4KiB. If the buffer size was then set to 16KiB, the expectation is that a single TCP session would not be able to utilize the 5Mb/s service. In this set of tests, we validate that expectation, and we additionally look at the effect when multiple simultaneous TCP sessions are sharing the upstream link.

In the second set of tests we correlate the buffer size and QoS of real-time applications during upstream self-congestion (i.e. saturation of the user's allotted bandwidth by a mix of applications). During congestion, packets from real-time applications may queue up in the buffer. If the buffer size is too large, this will increase the latency and may severely impact real-time applications such as online games and Voice over IP (VoIP). This set of tests examines whether changing the

buffer size can improve real-time applications' QoS or not.

The third set of tests examines the implications of buffer size when the token bucket feature defined in the DOCSIS 3.0 specification is utilized to provide the user a high burst transmit rate (as is currently the practice by several operators). In this test, the token bucket is set to 5MB and, after a suitable idle period, a 5MB file is transferred using a single TCP session (FTP). The TCP session runs through the process discussed earlier, beginning with slow start and transitioning to congestion avoidance. Since the token bucket size is equal to the size of the file, the rate shaping function in the CM doesn't come into play, and thus the limits to the transfer speed in this test are: a) the maximum speed of the upstream channel, which in this case is approximately 25Mb/s; and b) ability of a single TCP session to utilize the link in the presence of limited buffering.

We used upstream buffer sizes of 8KiB, 16KiB, 32KiB, 64KiB, 128KiB, and 256KiB, although in some test scenarios we limited the choice of buffer sizes due to time constraints. In the experiments the upstream bandwidth was provisioned to 1Mb/s, 2Mb/s, 5Mb/s, or 10Mb/s. These upstream speeds are indicative of services currently available in the market. The tests simulated RTTs of 40ms and 100ms. 40ms represents a local file transfer within a metropolitan network region (perhaps from a cache). 100ms represents a coast-to-coast session.

Network Setup

Figure 2 depicts the experimental setup, which emulates a typical home network behind a CM. The home network consisted of PCs running test applications and other hardware for measuring network characteristics and performance. Typical home networks would include a home gateway between the CM and the rest of the home network equipment. To eliminate any bias that might be introduced by the packet forwarding performance of a home gateway, we instead used a Gigabit switch to connect the home network equipment to the CM. We then configured the CM to allow multiple devices connecting to it.

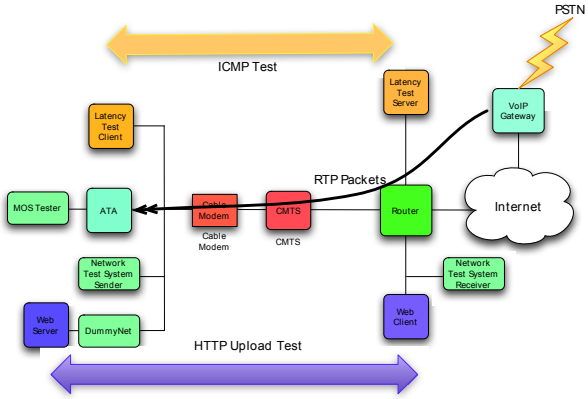

Figure 2

The CM was installed with the new DOCSIS 3.0 firmware that provides the capability of using the Buffer Control feature so that specific buffer queue sizes might be set. A CMTS with knowledge of such settings was connected to the CM. Inside the home network, there are four applications: (1) a PING test client, (2) a VoIP Analog Terminal Adaptor (ATA), (3) a Network Test System, and (4) a Web Server. The PING test client measured the RTT to the PING server in the test network. This was set up to validate the PING latency while the network was congested. The VoIP ATA was set up for the QoS test for real-time applications. A VoIP tester was connected to the ATA. This tester would measure the Mean Opinion Score (MOS) when the ATA made a phone call to the VoIP gateway in the Internet. The Network Test System allowed an easy way to simulate multiple TCP devices simultaneously. The Web server was setup to generate packets over TCP by uploading a large file via HTTP to the client behind the CMTS. In order to simulate the network

latency, Dummynet was used. Dummynet [18] is an internal packet filter for FreeBSD which can inject artificial delay in the network. It was connected between the CM and Web server. Delay was injected in both upstream and downstream. For example: we configured Dummynet to inject 20ms in both directions for the HTTP upload test to simulate 40ms RTT delay.

Baseline Results

As an initial test, we configured the CM with an upstream rate limit of 35Mb/s (effectively no rate-limiting), and investigated the achievable TCP data rate for a range of buffer sizes, both for the 40ms RTT case and for the 100ms RTT case. Figure 3 shows the resulting achieved data rate for 10 simultaneous upstream TCP sessions, along with the data rate that would be predicted based on the BDP. The maximum data rate that a single DOCSIS upstream channel could support is approximately 25Mb/s, so the achieved rates for 256KiB, 128KiB and perhaps 64KiB should be considered to be artificially limited by this constraint. It is interesting to note that in nearly all cases, when multiple TCP sessions were utilized, we were able to achieve a higher data rate (and in some cases a much higher data rate) than would be predicted based on the rule-of-thumb. Our conjecture is that the multiple TCP sessions were desynchronized (that is, the TCP sawtooths were not synchronized across sessions), and Appenzeller, Keslassy and McKeown [8] demonstrate that significantly less buffering than the bandwidth delay product is needed in this scenario.

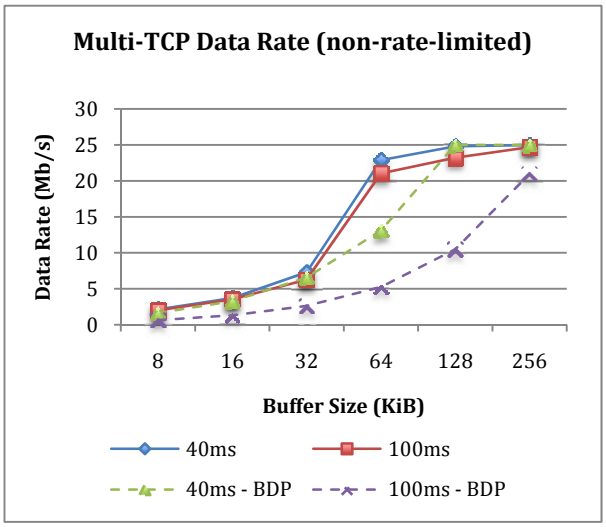

Figure 3

Test 1 Results

Figure 4 and Figure 5 show the results of data rate in various buffer sizes. Figure 4 shows the data rate of 1Mb/s and 2Mb/s upstream services. Figure 5 shows the data rate of 5Mb/s and 10Mb/s upstream services. The tests were executed with either 40ms or 100ms delay. All the tests were run with a single TCP stream.

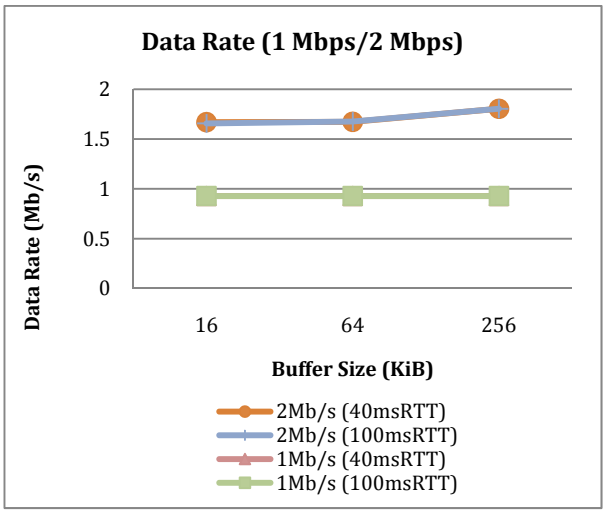

Figure 4

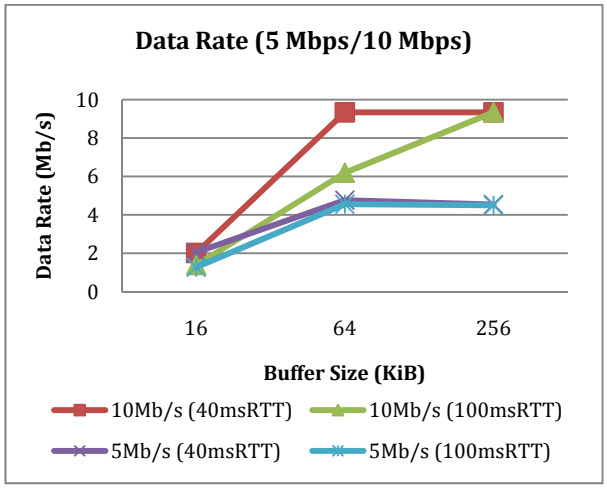

Figure 5

For 1Mb/s and 2Mb/s services, 16KiB buffer size was sufficient to utilize the upstream bandwidth with either 40ms and 100ms delay. For 5Mb/s, results showed that at least 64KiB buffer was needed when delay was 100ms. For 10Mb/s, results showed that 64KiB buffer was needed for 40ms delay and 256KiB buffer was needed for 100ms delay.

From the previous section, the tests performed with 10Mb/s upstream service for 16KiB and 64KiB showed significant bandwidth under-utilization. The question becomes: is it possible to achieve utilization of the whole bandwidth with multiple parallel TCP sessions? In additional experiments we used 10 TCP sessions established in parallel recording individual data rates and provided the aggregation plots in the graph provided in Figure 6a and Figure 6b for 10Mb/s upstream service with buffer sizes of 16KiB and 64KiB.

Figure 6a shows that with a 16KiB buffer, a single TCP session with 40ms delay is unable to achieve the full upstream rate, with only 1.97Mb/s of the 10Mb/s upstream service being used. By aggregating 10 TCP sessions and keeping other factors fixed, the channel utilization improved by a factor of four to 7.11Mb/s. Still, full utilization wasn't realized. When 64KiB buffer size was used, we saw no difference in throughput between a single TCP session and 10 TCP sessions. Both were able to achieve the full 10Mb/s rate limit.

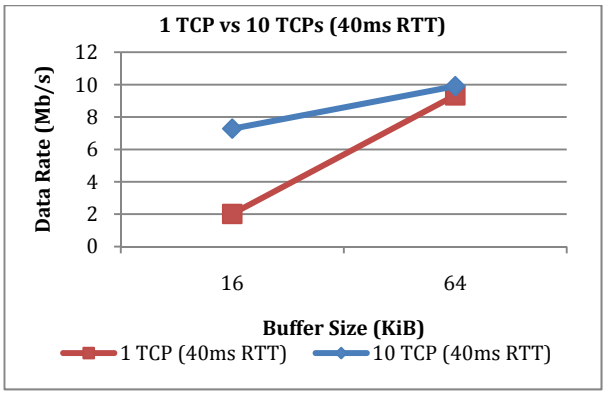

Figure 6a

Figure 6b shows further results with a 100ms delay. Bandwidth utilization could be improved for both 16KiB and 64KiB buffers when aggregating 10 TCP sessions. The test using 16KiB buffer size with a 100ms delay resulted in 4.95Mb/s upstream utilization, a three times improvement over the test that was conducted using a single TCP sessions. However, 5Mb/s of capacity remains unused. When using a 64KiB buffer size with 10 TCP sessions, there was improvement with a data rate of 8.77Mb/s compared to 6.05Mb/s for the single TCP flow test. Here again 1Mb/s of capacity was unused.

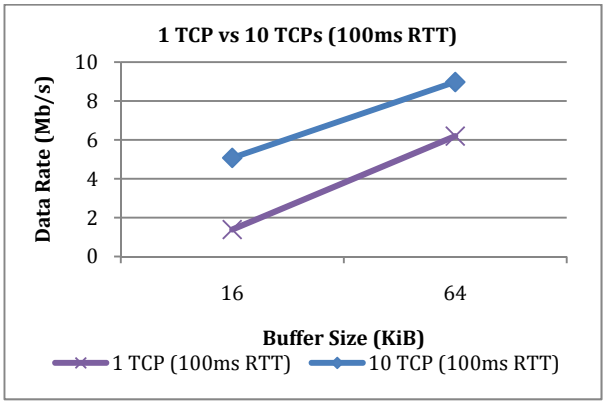

Figure 6b

Figure 7 and Figure 8 show the results of packet latency when an upstream link was

congested with a single TCP session. Figure 7 shows the latency of 1Mb/s and 2Mb/s upstream services. Figure 8 shows the latency of 5Mb/s and 10Mb/s upstream services when the upstream link was congested with a single TCP session.

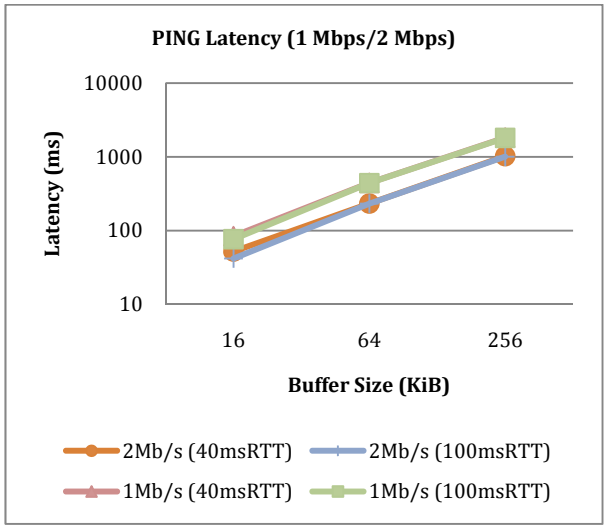

Figure 7

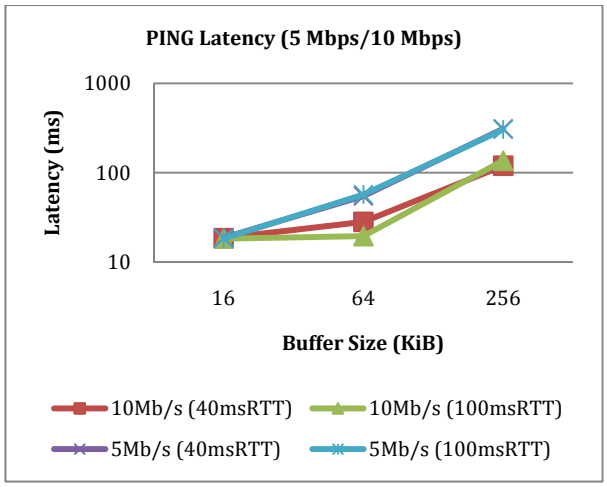

Figure 8

For all upstream services, the pattern is that the bigger the buffer, the higher the PING latency. The observed pattern of the test results show (as expected) that the latency is directly proportional to the buffer size. If the buffer size is increased to four times the size, the latency would increase roughly four times. For example, when testing with a 2Mb/s upstream service, the PING latency was 52.04ms for a 16KiB buffer. The PING latency increased to 200ms for a 64KiB buffer and 1019.17ms for a 256KiB buffer. However, this pattern was not observed when using a 10Mb/s service for 16KiB and 64KiB buffer sizes. This is because a single TCP session was unable to utilize the full 10Mb/s service with 40ms and 100ms delay. Therefore, the PING packets proceeded without congestion impact.

Test 2 Results

In this test, we wanted to understand how buffer size would impact the VoIP MOS under congestion. The VoIP ATA was connected into the simulated home network in our test environment. We started a large upstream file transfer (using a single TCP session) to simulate a self-congested upstream. The Dummynet test set injected either 40ms or 100ms round-trip latency to the TCP stream. The two latency values impact the dynamics of the TCP congestion avoidance algorithm differently, and thus result in a different pattern of CM buffer usage. For each bandwidth/buffer/delay permutation, we ran 10 tests and computed the average MOS.

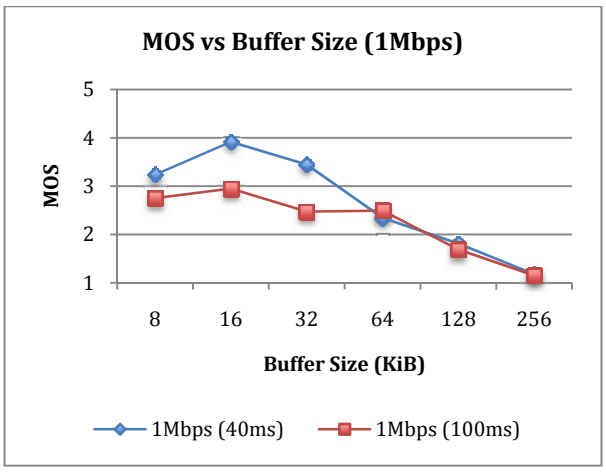

Figure 9a

Figure 9a shows the results when the upstream was configured to 1Mb/s. In this case, the 16KiB buffer gave the best MOS result regardless of the TCP RTT. When the buffer size increased to 64KiB, the MOS started to drop. In fact, when the buffer size was set to 256KiB with 100 ms delay, many tests couldn't be completed. The test set reported a Timeout. The timeout could be caused by long delay introduced by the large buffer.

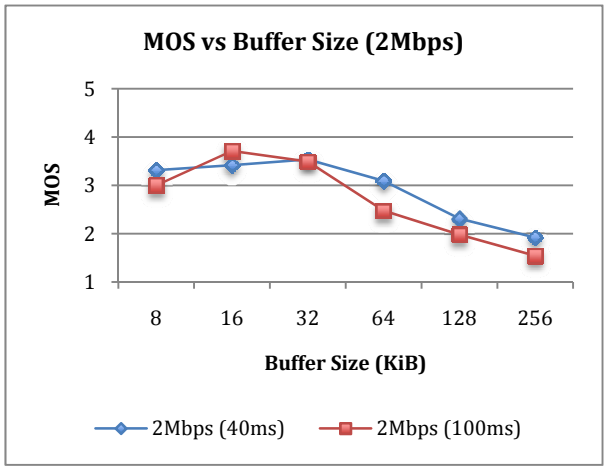

Figure 9b

Similar results were obtained for 2MB/s, as shown in Figure 9b.

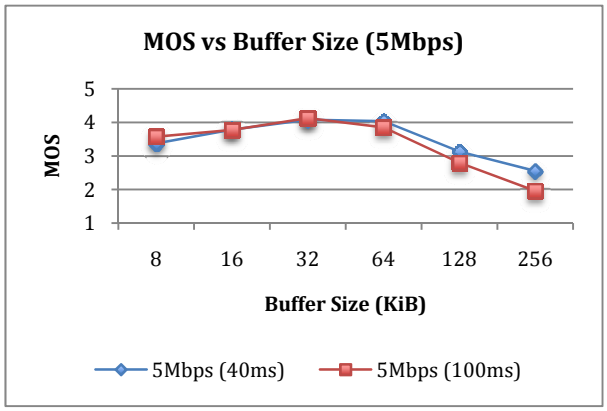

Figure 9c

Figures 9c and 9d show the results of 5Mb/s and 10Mb/s upstream bandwidths. For both of these cases, the best MOS results were recorded when the buffer size was set between 32KiB and 64KiB. What was interesting is when the buffer was set below 64KiB, there should not be any congestion because the buffer size was too small for the competing TCP session to utilize the full upstream bandwidth. Also when the buffer size was set to 8KiB or 16KiB, the VoIP traffic should see very low buffering latency (between 6ms and 26ms). Both of these factors should result in high MOS scores. However, the MOS score was degraded when the buffer size was set to 8KiB or 16KiB, and this low MOS is likely to be caused by packet loss due to the periodic saturation of the small CM buffer by the competing TCP congestion window.

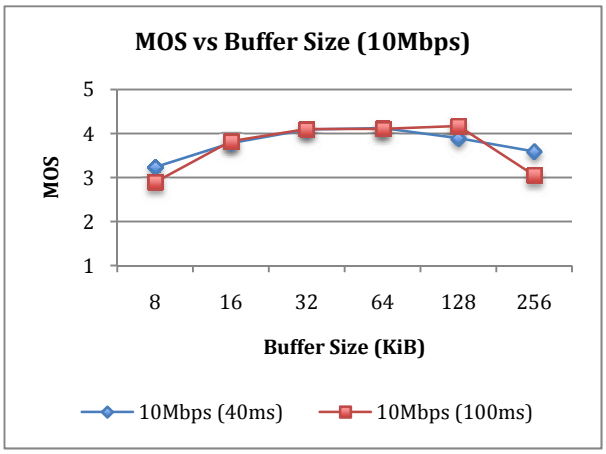

Figure 9d

When we increased the buffer size above 128KiB, the MOS started to deteriorate. In fact, the test set failed to complete some tests and required us to re-run the test when we set the upstream bandwidth to 10Mb/s with 256KiB buffer and 100ms delay. This could be caused by the long delay of the VoIP signaling packets for the call setup.

Another interesting observation is that, while adding latency to the TCP session did in some cases affect the VoIP MOS, it didn't point to a different optimal buffer size.

Test 3 Results

In this test, the CM's rate-shaping token bucket depth was set to 5MB to mimic the configuration that some operators use to boost performance for bursty interactive traffic. The test set was then used to transit a 5MB file in the upstream direction using FTP. The test measured the transfer time and, from this, an average data rate was calculated. Due to the fact that the file size was equal to the token bucket depth, there was effectively no upstream rate-shaping in play. Figure 10 shows the result of Average Transfer Time for the scenario with 40ms RTT and the scenario with 100ms RTT.

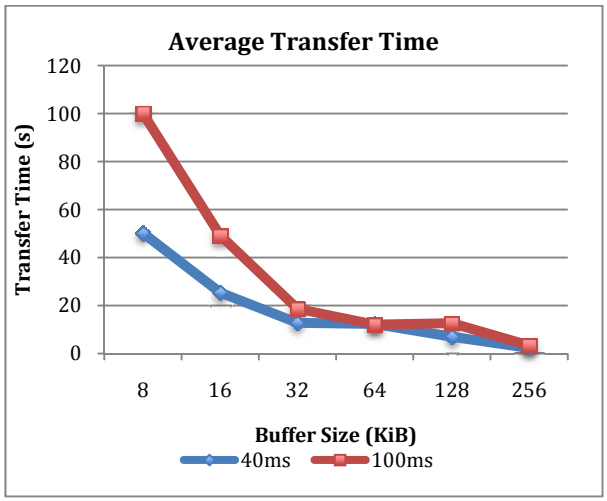

Figure 10

Figure 11 shows the calculated average throughput based on the average transfer time and the file size. It is clear that the choice of buffer size can have a dramatic impact on achievable throughput during this performance boost period. It is important to note that the average throughput shown in Figure 11 is for the entire file transfer, and so includes the slow-start effect as well as the initial FTP hand-shaking.

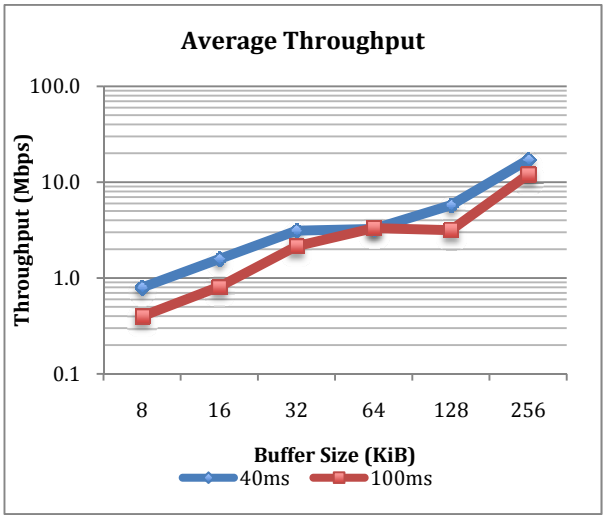

Figure 11

FUTURE WORK

Additional work is necessary to more fully understand the impact of CM buffering on application performance and user experience. The overarching goal of this work is to enable cable operators to optimize network performance and user experience across a wide range of application and usage scenarios.

The experimental work described in this paper focused both on upstream bulk TCP performance, and on the impact that bulk TCP traffic would have on a VoIP session across a range of buffer sizes in fairly isolated and controlled scenarios. Further work will seek to examine the effect of buffering on some more real-world application scenarios, such as a scenario in which TCP sessions are numerous and short-lived, such that many of them stay in the slow-start phase for their entire duration, and never (or rarely) enter the congestion avoidance phase. Additionally, future work should assess the impact of buffer sizing on other latency or loss sensitive applications beyond VoIP.

CONCLUSIONS

We have shown that the size of the upstream service flow buffer in the cable modem can have significant effects on application performance. For applications that use TCP to perform large upstream transmissions (i.e. upstream file transfers such as uploading a large video clip), insufficient buffering capacity can limit throughput. Applications utilizing a single TCP session see the most pronounced effect. Applications that are latency sensitive, on the other hand, see degraded performance when too much buffer capacity is provided.

Cable modems deployed today appear to have significantly greater buffering than is needed to sustain TCP throughput, potentially causing high latency when the upstream is congested. The result is poorer than expected application performance for VoIP, gaming, Web surfing, and other applications that are latency-sensitive.

A new feature for DOCSIS 3.0 CMs allows operators to configure the upstream buffer size for each upstream service flow in order to optimize application performance and to improve user experience. In choosing the buffer size, the operator will need to consider the upstream QoS parameters for the service flow, the expected application usage for the flow, as well as the service goals for the flow.

Service features that utilize a large token bucket size (in order to provide high throughput for short bursts) complicate matters since the buffer size cannot realistically be resized in real time. Thus a buffer configuration that may be optimized to provide a good balance in performance between TCP uploads and real-time services for the configured sustained traffic rate, may result in poorer than expected burst speeds.

Further work is needed to evaluate the performance impact that buffer size has on a wider range of application scenarios.

REFERENCES

[1] V. Jacobson, Congestion Avoidance and Control, ACM SIGCOMM '88, August 1988.

[2] J. Postel, Transmission Control Protocol, STD 7, RFC793, September 1981.

[3] Braden, B., et al., Recommendations on Queue Management and Congestion Avoidance in the Internet, RFC 2309, April 1998.

[4] S. Floyd, Congestion Control Principles, BCP 41, RFC 2914, September 2000.

[5] K. Ramakrishnan, S. Floyd and D. Black, The Addition of Explicit Congestion Notification (ECN) to IP, RFC 3168, September 2001.

[6] M. Allman, V. Paxson and E. Blanton, TCP Congestion Control, RFC 5681, September 2009.

[7] C. Villamizar and C. Song, High Performance TCP in ANSNet. ACM CCR, 24(5):45-60, 1994.

[8] G. Appenzeller, I. Keslassy, and N. McKeown, Sizing Router Buffers, ACM SIGCOMM, USA, 2004.

[9] M. Dischinger, et al., Characterizing Residential Broadband Networks, Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, Oct. 24-26, 2007, San Diego, CA, USA.

[10] A. Vishwanath, V. Sivaraman, M. Thottan, Perspectives on Router Buffer Sizing: Recent Results and Open Problems, ACM CCR, 39(2):34-39, 2009.

[11] ITU-T Recommendation G.114, One-way Transmission Time, http://www.itu.int

[12] ITU-T Recommendation G.107, The E-model, a computational model for use in transmission planning, http://www.itu.int

[13] ITU-T Recommendation G.108, Application of the E-model: A planning guide, http://www.itu.int

[14] ITU-T Recommendation G.109, Definition of categories of speech transmission quality, http://www.itu.int

[15] R. G. Cole and J. H. Rosenbluth, Voice over IP performance monitoring, *ACMCCR,* 31(2), 2001.

[16] Internetworking lectures from La Trobe University, http://ironbark.bendigo.latrobe.edu.au/subjects/INW/lectures/19/, 2011.

[17] J. Gettys, Bufferbloat: Dark Buffers in the Internet, http://mirrors.bufferbloat.net/Talks/PragueIETF/IETFBloat7.pdf, April 2011.

[18] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM Computer Communication Review, Vol. 27, No. 1, pp. 31-41, January 1997.