NetScaler TCP Optimization

TCP Optimization in NetScaler can can be integrated with other network services on carrier-grade appliances or can be deployed as dedicated virtual network function on COTS hardware.

Citrix NetScaler is a leader for TCP optimization in mobile networks. Citrix has more than 15 years of experience in this space with a large number of deployments with top tier operators around the world.

In this document, we will first discuss the TCP performance gap in mobile operators and then provide an in-depth overview of TCP congestion control technologies and Citrix NetScaler TCP Optimization solutions. In particular we introduce TCP Nile, a new high-performance TCP congestion algorithm specifically designed by Citrix for wireless networks.

TCP Performance Gap in Mobile Networks

TCP congestion control technologies were originally developed in the early days of the Internet to deal with congestion in the Internet. These technologies performed well for fixed-line networks. However, the transition to using the same techniques in mobile networks hasn't been quite so simple. The mobile network has proven a far more challenging arena for TCP protocols when it comes to achieving the same success.

This is partly because a mobile network throws up several situations and challenges not often seen in fixed-line networks. The available bandwidth in a modern mobile network changes far more frequently because of variables such as environmental factors and mobility.

The multitude of technologies (GRPS, CDMA, UMTS, LTE), generations (2.5G, 3G, 3.5G, 4G), variations (EDGE, EVDO, HSPA, LTE-A), and mobile terminal classes, in conjunction with the different requirements of various applications (web browsing, large file downloads, video streaming, real-time communications, etc.) result into an enormous number of combinations and a highly variable and mixed environment.

The mobile network has some unique characteristics that can also affect the performance of TCP protocols:

- The combination of high throughput with relatively high latency results in network paths that suffer from the same effects as long/fat networks (LFN). Many content servers are not properly tuned for such conditions.

- The two directions of data transmission (downstream and upstream) may have different bandwidth limits and transmission delays. Data transmission is asymmetric, in that sense.

- There are random packet losses that are not necessarily caused by congestion but as a result of radio interference or handover between cells. Packet losses occasionally can take place in bursts.

- Both UMTS and LTE provide schemes for frame recovery and retransmission at the lower layers of the stack. For example, the RLC layer implements ARQ (Automatic Report reQuest) and the MAC layer employs HARQ (Hybrid ARQ). Such link-layer operations may have undesirable impacts on the TCP latency estimation.

- One of the new capabilities of HSPA and LTE-Advanced is carrier aggregation. Most HSPA and LTE networks are expected to employ this (some already have). The first round of LTE-Advanced deployments, is that carrier aggregation introduces additional unpredictability in end-to-end bandwidth and latency.

Table 1: TCP Performance Gap

Location Signal Condition {RSRP, SINR} values represent signal strength and signal/noise ratio	Excellent {-64, 27}	Good {-90, 27}	Fair {-110, 15}
Maximum Achievable UDP Goodput (Mbps)	81	55	50
Actual TCP Throughput (Mbps)	34	30	27
TCP Speedup Potential	138%	83%	85%

As a result, TCP protocols often cannot achieve the expected performance that the underlying network can support. Table 1 illustrates the TCP performance gap in mobile networks. In this study [1] of a LTE network, TCP throughput was measured by downloading large files in several locations with different signal strength and signal/noise ratio. The maximum achievable application layer UDP throughput (i.e., Goodput) was also recorded. The results show that TCP achieves substantially lower throughput than what the network is capable of supporting. There is significant potential to apply TCP optimization technologies to speed up TCP performance in mobile networks and close the gap.

Comparison of TCP Algorithms

In this section, we will present an overview of the key TCP concepts and compare the various TCP algorithms in the context of mobile networks.

TCP Congestion Control Framework

TCP congestion control consists of a set of algorithms that are key to the performance of TCP protocol. Over the last 30 years, a large number of TCP algorithms have been developed. We will cover a number of TCP algorithms that are widely deployed today on the Internet.

TCP algorithms use a mechanism called "windowing" to regulate the amount of unacknowledged traffic that a sender can transmit to a receiver. The majority of TCP algorithms are based on the "additive-increase, multiplicative-decrease" principle: TCP increases the congestion window in a relatively conservative, additive fashion, but aggressively decreases it, in a multiplicative fashion, when it detects congestion. There are four basic aspects of TCP congestion control algorithms that re fundamental:

•Slow Start
•Congestion Avoidance
•Fast Retransmit
•Fast Recovery

They provide a framework that makes sure that the Internet is protected from congestion collapse, competing TCP flows are handled with fairness, and end-to-end performance is optimized for throughput, delay and loss.

Congestion control algorithms

Table 2 summarizes the main TCP congestion control algorithms that have been introduced over the last 30 years. Many of the algorithms are considered "Experimental". The two algorithms CUBIC and Compound are used extensively today on Linux and window servers.

Table 2: Summary of TCP algorithms

TCP variant	Base	Year	Update features
Citrix Nile	Citrix Proprietary	2012	Designed for mobile networks
CUBIC	BIC	2008	The cwnd control as a cubic function of time elapsed since a last congestion event.
YeAH	STCP, Vegas	2007	Switching b/w fast (STCP) and slow (New Reno) mode depending on a combined Vegas-type and DUAL-type estimate, precautionary decongestion
Illinois	NewReno, DUAL	2006	AIMD factors as functions of the queuing delay
Compound	HS-TCP, Vegas	2005	Two components (slow and scalable) in the cwnd calculation.
BIC	HS-TCP	2004	Binary cwnd search, limited SS
Hybla	New Reno	2004	Scaling the increase steps in SS and CA to the reference RTT, Data packet pacing, initial ssthresh estimation.
H-TCP	New Reno	2004	Cwnd increase steps as a Function of time elapsed since the last packet Loss detection, scaling increase step to a reference RTT, multiplicative decrease coefficient adaptation.
Westwood+	Westwood	2004	Estimate of available bandwidth (RTT Granularity)
FAST	Vegas	2003	Constant-rate cwnd equation-based update.
HS-TCP	New Reno	2003	Additive increase steps and multiplicative decrease factors as functions of the congestion window size, limited SS.
Veno	New Reno Vegas	2002	Reno-type CA and FR increase/decrease coefficient adaptation based on bottleneck buffer state estimation
LP	New Reno	2002	Early congestion detection
Westwood	New Reno	2001	Estimate of available bandwidth (ACK granularity), FR
DSACK	SACK	2000	Reporting duplicate segments
Eifel	New Reno	2000	Differentiate between transmitted and retransmitted data packets.
New Reno	Reno	1999	FR resistant to multiple losses
SACK	RFC793	1996	Extended information in feedback messages
FACK	Reno, SACK	1996	SACK-based loss recovery algorithm
Vegas	Reno	1995	Bottleneck buffer utilization as a primary feedback for the CA and secondary for the SS
Reno	Tahoe	1990	FR
Tahoe	RFC793	1988	SS, CA, FR

SS=Slow Start, CA=Congestion Avoidance, FR=Fast Recovery.

Basic principles

To avoid congestion collapse, TCP uses a multi-faceted congestion-control strategy. For each connection, TCP maintains a congestion window, referred to herein as cwnd, limiting the total number of unacknowledged packets that may be in transit end-to-end.

TCP uses a mechanism called slow start to increase the congestion window after a connection is initialized and after a timeout. It starts with a window size which is a number of times the maximum segment size (MSS), called the initial congestion window. Although the initial rate is low, the rate of increase is actually very rapid: For every packet acknowledged, the congestion window increases by 1 MSS so that the congestion window effectively doubles for every round-trip time (RTT). When the congestion window exceeds a threshold ssthresh the algorithm enters a new state, called congestion avoidance. In most implementations, the initial ssthresh is large, and so the first slow start usually ends after a loss. However, ssthresh is updated at the end of each slow start, and will often affect subsequent slow starts triggered by timeouts.

Timeouts correspond to cases an acknowledgement for a packet wasn't received within the RTO (retransmission timeout) interval. In congestion avoidance phase, as long as non-duplicate ACKs are received, the congestion window is additively increased by one MSS every RTT. When a packet is lost, the likelihood of duplicate ACKs being received is very high. Congestion control algorithms are typically classified based on how they calculate the cwnd and ssthresh during the congestion avoidance phase, in response to a series of duplicate ACKs, or reacting to timeouts.
We will briefly discuss some of the popular TCP algorithms that we support on NetScaler. We will cover the details of TCP Nile – a Citrix proprietary TCP algorithm in the next section.

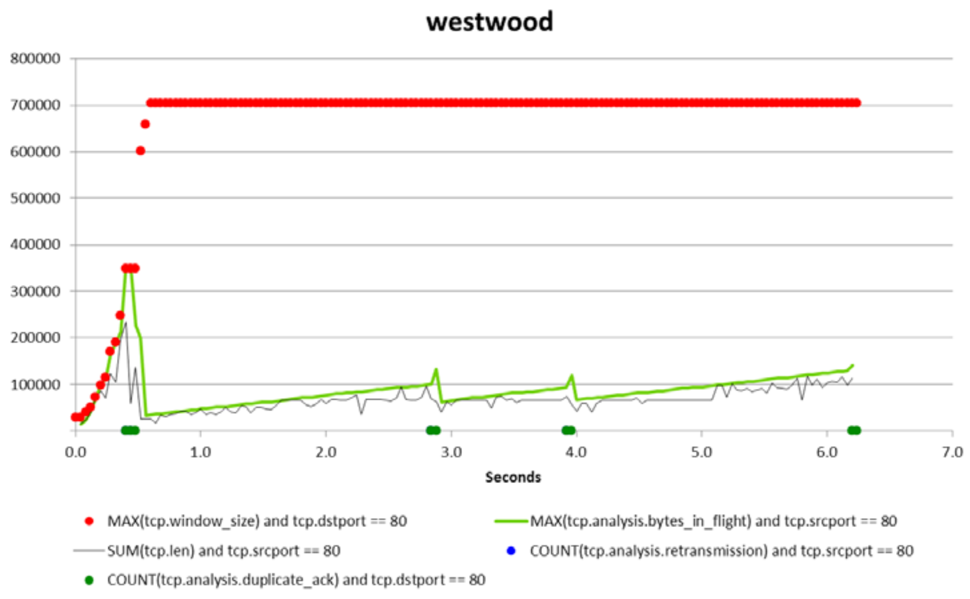


Figure 1: Westwood+ downloading 10MB file (example)

Westwood+

TCP Westwood+ is a sender-side only modification of the TCP Reno protocol stack that optimizes the performance of TCP congestion control over both wire-line and wireless networks. TCP Westwood+ is based on end-to-end bandwidth estimation to set congestion window and slow start threshold after a congestion episode, that is, after three duplicate acknowledgments or a timeout. The bandwidth is estimated by properly low-pass filtering the rate of returning acknowledgment packets.

The rationale of this strategy is simple: in contrast with TCP Reno, which blindly halves the congestion window after three duplicate ACKs, TCP Westwood+ adaptively sets a slow start threshold and a congestion window which takes into account the bandwidth used at the time congestion is experienced. TCP Westwood+ significantly increases throughput over wireless links and fairness compared to TCP Reno/New Reno in wired networks.

TCP Westwood+ is an evolution of TCP Westwood. The main novelty of Westwood+ was the algorithm used to estimate the available bandwidth end-to-end. In fact, it was soon discovered that the Westwood bandwidth estimation algorithm did not work well in the presence of reverse traffic due to ACK compression. This phenomenon was explained in terms of aliasing effects.

BIC

Binary Increase Congestion (BIC) control is an implementation of TCP with an optimized congestion control algorithm for high speed networks with high latency. BIC has a unique congestion window (cwnd) algorithm. This algorithm tries to find the maximum size to keep the window at for a long period of time, by using a binary search algorithm.

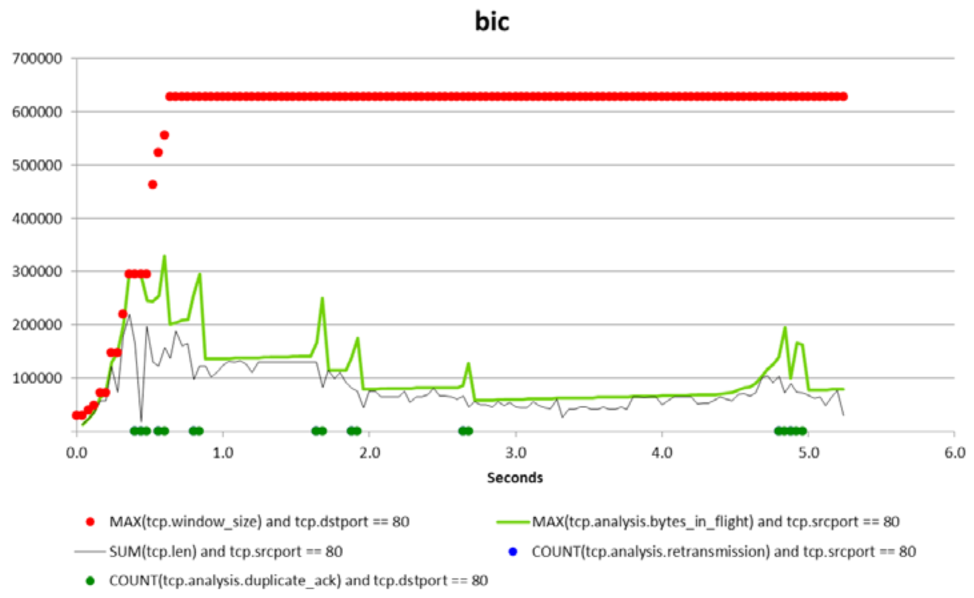


Figure 2: TCP BIC downloading 10MB file (example)

CUBIC

CUBIC is a less aggressive and more systematic derivative of BIC, in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event.

Being a cubic function, there are two components to window growth. The first is a concave portion where the window quickly ramps up to the window size before the last congestion event. Next is the convex growth where CUBIC probes for more bandwidth, slowly at first then very rapidly. CUBIC spends a lot of time at a plateau between the concave and convex growth region which allows the network to stabilize before CUBIC begins looking for more bandwidth.

Another major difference between CUBIC and standard TCP flavors is that it does not rely on the receipt of ACKs to increase the window size. CUBIC's window size is dependent only on the last congestion event. With standard TCP, flows with very short RTTs will receive ACKs faster and therefore have their congestion windows grow faster than other flows with longer RTTs. CUBIC allows for more fairness between flows since the window growth is independent of RTT.

Qualitative performance comparison

In Table 3 we rate the performance of the supported congestion control algorithms, in a qualitative and approximate fashion, in terms of:

- Throughput – Whether the algorithm can achieve high throughput quickly in ideal conditions
- Random Packet loss – How effectively the algorithm responds to random, non-correlated packet loss
- Congestion – How the algorithm reacts to correlated packet loss resulting from congestion
- Friendliness – How friendly is the algorithm in sharing a common network path with other algorithms
- Fairness – How fairly it shares network resources between multiple competing flows of different RTTs
- Queuing Delay – Whether the aggressiveness of the algorithm leads to large queuing delays
- Bufferbloat – Whether the algorithm exacerbates the bufferbloat caused by the RLC buffers

Table 3: Qualitative performance of select congestion control algorithms

Algorithm	Throughput	Random Packet Loss	Congestion	Friendliness	Fairness	Queuing Delay/ Bufferbloat
Citrix Nile	5	5	5	5	5	5
New Reno	3	3	4	4.5	3	4
Westwood+	4	4	3	4.5	3	5
CUBIC	4.5	3.5	5	4	5	3.5
BIC	5	4.5	4.5	3	4.5	3
Illinois	4.5	5	4.5	4	5	5

NetScaler TCP optimization Solution

NetScaler TCP optimization is a solution specifically designed to help mobile operators to improve user experience. It is built on top of the high performance NetScaler ADC platform with its unique high-parallelism and ultra-low latency packet engine architecture that maximizes the performance of multi-core processors. Figure 3 outlines key benefits of the NetScaler solution.

In TCP Optimization, NetScaler acts as a transparent 'relay' between the mobile network and the Internet. On the Internet side, it uses TCP algorithms specifically formulated to address the challenges in mobile networks. The TCP proxy receives data as soon as possible from the content server on the Internet and then attempts to deliver this data to the handset in a way that matches the current network conditions in the most adaptive fashion possible. With NetScaler TCP optimization, data connection speeds can be dramatically improved, resulting in a higher QoE for the subscriber.

NetScaler TCP optimization brings two key benefits to the mobile subscribers:

- The content servers on the Internet may run different TCP stacks. Therefore, the latency, bandwidth, and packet loss can vary widely depending on where the servers are and how they are connected. The Traffic Manager acts as a relay, shielding mobile subscribers from the diversity of the content servers and network characteristics on the Internet side.
- NetScaler uses optimal TCP settings for the mobile network in order to provide quick adaptation to changing network conditions and speed up file downloads and web browsing when network resources are available.

Act as a relay between wireless & wireline	Shield mobile subscribers from the diversity of content servers and network characteristics
Deliver best TCP Performance for mobile	Optimize mobile TCP settings to respond quickly to changing local traffic conditions
Reduce contention among subscribers	Ensure consistent performance across all TCP connections regard-less of their destination
Tune parameters based on advanced policy	Fine-tune TCP performance based on advanced Policy Manager capabilities

Figure 3: TCP Optimization Benefits

NetScaler separates the wireless-side network and the Internet-side network and enables you to apply an optimal set of TCP parameter settings for each network. For example, to provide full server-side network speed, the NetScaler advertises a receiving window that takes into account the characteristics of the server-side network. In addition, when needed, the NetScaler applies flow control on the server-side when the client-side network cannot sustain the server-side sending rate.

On the client-side, NetScaler uses optimized congestion control algorithms, fine-tuned TCP send buffers, and other specialized TCP settings intended for the characteristics of the wireless network.

Figure 4 shows how NetScaler enables higher throughput for "Long Fat Networks" with random packet loss.

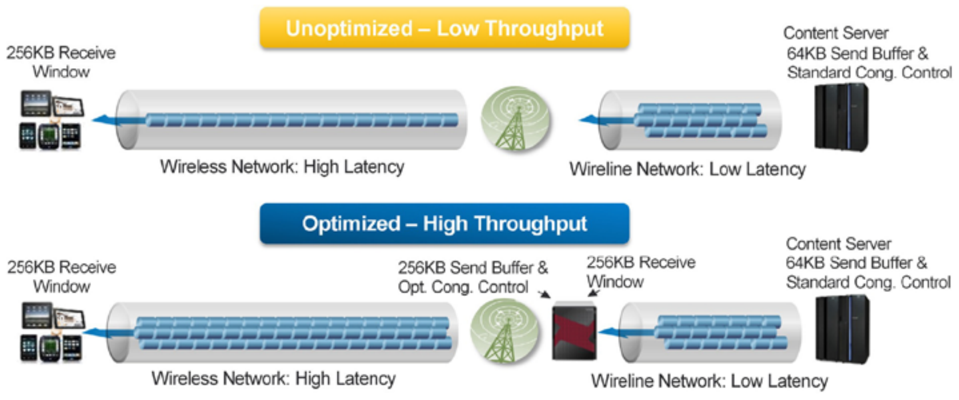

Unoptimized – Low Throughput
256KB Receive Window
Content Server
64KB Send Buffer & Standard Cong. Control
Wireless Network: High Latency
Wireline Network: Low Latency
Optimized – High Throughput
256KB Receive Window
256KB Send Buffer & Opt. Cong. Control
256KB Receive Window
Content Server
64KB Send Buffer & Standard Cong. Control
Wireless Network: High Latency
Wireline Network: Low Latency

Figure 4: TCP Optimization Overview

The un-optimized network provides overall low throughput due to the high latency wireless network. With NetScaler TCP optimization, higher throughput is provided by adjusting the TCP receive window size to match the window size of the wireless device and providing optional congestion control configuration parameters.

NetScaler enables two kinds operator-defined TCP optimization parameters:

- **Global TCP optimization** — NetScaler enables the configuration of global TCP optimization parameters. The global TCP optimization parameters take effect system-wide. These parameters are applied to all connections handled by NetScaler regardless of network type or other conditions, such as network latency or time of day.

- **Policy-based TCP optimization** — NetScaler offers a feature called policy-based TCP optimization. This feature uses TCP optimization profiles to apply TCP optimization dynamically to individual connections. A TCP optimization profile contains a set of TCP parameters that tune TCP for the particular NetScaler deployment (for example, for a 3G or LTE network). The profile is applied to connections based on policy input, such as PCRF or RADIUS attribute values. For more information.

Key TCP optimization features
Key optimization features that can be enabled by global TCP configuration or profiles are:

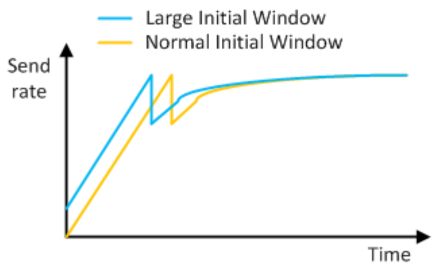
- TCP Congestion Control optimized for high speed wireless networks — The following algorithms are currently supported:
  - -Westwood+
  - -BIC
  - -CUBIC
  - -Nile

- Selective Acknowledgment (SACK) — Improves the performance of data transmission, especially in long fat networks (LFNs).

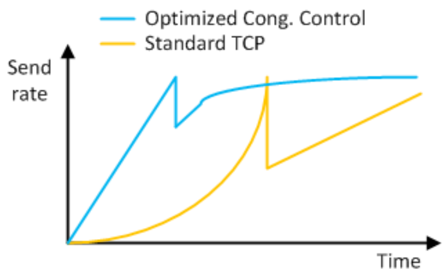
- TCP Window Scaling — Allows efficient transfer of data over long fat networks (LFNs).

- Configurable Buffer Sizes — In today's networks, you need to be able to adjust the TCP send and receive buffer sizes to minimize data transfer interruption when too much data is sent over the network. If the buffers are too small, the corresponding windows may be occasionally overrun. This can result in the flow control mechanisms stopping the data transfer until the buffer is empty. In NetScaler, TCP profile configuration options allow the adjustment and tuning of send/receive buffer sizes.

- Large Initial Congestion Window — To avoid causing network congestion, TCP flows use a slow-start method that avoids sending more data than the network is capable of transmitting. At the start of each connection, the number of segments transmitted increases until packet loss occurs or another threshold is reached. For performance-tuning purposes, NetScaler enables you to adjust and optimize the size of the initial congestion window.



- Detection of spurious retransmissions and response with the appropriate action — The objective is to avoid unnecessary congestion control measures. This is particularly beneficial on wireless networks with variable latency (RTT).



- TCP Keep-Alive — Checks the operational status of the peers at specified time intervals to prevent the link from being broken.

Standards and specifications
The NetScaler TCP optimization implementation has excellent capabilities of automatically adapting to wireless network conditions and detecting/avoiding congestion. The summarized list of RFCs and IETF (Draft) Standards supported by the TCP/IP stack is provided in Table 4 on the following page. The list is not exhaustive: Primarily the RFCs that have relevance to performance are included in this table.

Table 4: List of performance-related RFCs supported by the TCP/IP stack

RFC	Title	Obsoleted by
896	Congestion Control in IP/TCP Internetworks (Nagle)[1]	
1122	Requirements for Internet Hosts — Communication Layers	
1191	Path MTU Discovery	
1981	Path MTU Discovery for IP version 6	
1323	TCP Extensions for High Performance	RFC 7323
2018	TCP Selective Acknowledgment Options	
2581	TCP Congestion Control	RFC 5681[2]
2883	An Extension to the Selective Acknowledgement (SACK) Option for TCP	
2914	Congestion Control Principles	
2988	Computing TCP's Retransmission Timer	RFC 6298
3168	The Addition of Explicit Congestion Notification (ECN) to IP	
1353	Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations	
3390	Increasing TCP's Initial Window	RFC 6928
3782	The NewReno Modification to TCP's Fast Recovery Algorithm	RFC 6582

The NetScaler software supports the following TCP capabilities:

- Defending TCP against spoofing attacks. The NetScaler implementation of window attenuation is [RFC 4953] compliant.
- Explicit Congestion Notification (ECN), which sends notification of the network congestion status to the sender of the data and takes corrective measures for data congestion or data corruption. The NetScaler implementation of ECN is [RFC 3168] compliant.
- Round Trip Time Measurement (RTTM) using the TimeStamp option. For the TimeStamp option to work, at least one side of the connection (client or server) must support it. The NetScaler implementation of TimeStamp option is [RFC 1323] compliant.
- Detection of spurious retransmissions can be done using TCP duplicate selective acknowledgement (D-SACK) and forward RTO-Recovery (F-RTO). In case of spurious retransmissions, the congestion control configurations are reverted to their original state. The NetScaler implementation of D-SACK is [RFC 2883] compliant, and F-RTO is [RFC 5682] compliant.
- Congestion control using New Reno, BIC, CUBIC, Nile and Westwood algorithms.
- Window scaling to increase the TCP receive window size beyond its maximum value of 65,535 bytes.
- TCP maximum congestion window size that is user configurable. The default value is 8190 bytes.
- Selective acknowledgment (SACK), using which the data receiver (either NetScaler software or a client) notifies the sender about all the segments that have been received successfully.
- Forward acknowledgment (FACK) avoids TCP congestion by explicitly measuring the total number of data bytes outstanding in the network, and helping the sender (either a NetScaler ADC or a client) control the amount of data injected into the network during retransmission timeouts.
- TCP connection multiplexing enables reuse of existing TCP connections. NetScaler stores established TCP connections to the reuse pool. Whenever a client request is received, NetScaler checks for an available connection in the reuse pool and serves the new client if the connection is available. If it is unavailable,

[1] The current recommendation is to disable the TCP Nagle algorithm, since it has been found to not be optimal for web browsing use cases at HSDPA/LTE speeds.

[2] [RFC 5681] recommends against the [RFC 3465] capability of increasing MSS by more than 1.

NetScaler creates a new connection for the client request and stores the connection to the reuse pool. NetScaler supports connection multiplexing for HTTP, SSL, and DataStream connection types.

- Dynamic receive buffering allows the receive buffer to be adjusted dynamically based on memory and network conditions.

- MPTCP connections between client and NetScaler. MPTCP connections are not supported between NetScaler and the backend server. The NetScaler implementation of MPTCP is [RFC 6824] compliant.
  - For MPTCP to work, both sides of the connection (client and server) must support it. If you use NetScaler as an MPTCP gateway for your servers, the servers do not have to support MPTCP.
  - NetScaler does not initiate subflows (MP_JOIN's), expecting the client to initiate subflows.

- TCP keep-alive to monitor the TCP connections to verify if the peers are up.

Additionally, NetScaler provides configuration support for the following:

- TCP segmentation offload.

- Synchronizing cookie for TCP handshake with clients. Disabling this capability prevents SYN attack protection on NetScaler.

- Learning MSS to enable MSS learning for all the virtual servers configured on NetScaler

Global TCP Optimization
Global TCP optimization configures TCP settings - to provide system-wide TCP optimization (that is, for all connections being handled by the Traffic Manager). The global TCP optimization settings are always in effect but may be overridden by an active TCP optimization profile.

Initial Congestion Window
To avoid causing network congestion, TCP flows use a slow-start method that avoids sending more data than the network is capable of transmitting. At the start of each connection, the number of segments transmitted (window size) increases until packet loss occurs or another threshold is reached.

The initial congestion window has been set to 4 segments as part of the TCP standard. This is too conservative for today's networks, and it is putting an upper bound on the performance of web browsing use cases. NetScaler sets this by default to 16 segments. For global performance-tuning purposes, Citrix enables you to alter the size of the initial congestion window by using CLI commands.

TCP Buffer Sizes
With client-server connections, the client tells the server the number of bytes it is willing to receive at one time from the server. This is the client's receiving window. Likewise, the server tells the client how many bytes of data it is willing to take from the client at one time. This is the server's receiving window.

The TCP receive buffer size defines the maximum size of the receiving window. If the receive window size is too small, the receive window buffer may be frequently overrun.
This can result in the flow control mechanism stopping the data transfer until some space becomes available in the receive buffer.
The TCP send buffer size is used by the sender to store the data to be sent on the network. If the send buffer is too small, it could limit the amount of data to be injected on the network. The send buffer, therefore, needs to be large enough to be able to fill the network by sending data.

The TCP read-ahead-gap (RAG) specifies the maximum number of kilobytes that can be read from the server before sending data to the client and vice-versa. Increasing the RAG can increase the overall throughput of the system.

Configuration options in the CLI allow you to adjust TCP buffer sizes globally.

Global TCP Configuration Options
Several additional TCP parameters can be configured on a global basis through the CLI and Web GUI, including:

- Selection of a TCP congestion control algorithm[3]:
  - reno — TCP New Reno
  - westwood — TCP Westwood+ (system default)
  - bic — Binary Increase Congestion (BIC) control
  - cubic — CUBIC TCP
  - nile — TCP Nile (Citrix-proprietary)

- Configurable TCP parameters, including:
  - Wireless- and Internet-side TCP send and receive buffer sizes
  - Maximum Keep-Alive Idle Timeout
  - Keep-Alive Probe Number
  - Keep-Alive Probe Interval
  - TCP FIN Timeout
  - Advertised Maximum Segment Size
  - Maximum Congestion Window
  - Initial Congestion Window
  - Initial-Receive-Window

- Options for enabling and disabling protocol features:
  - TCP Keep-Alive
  - Maximum Packet Re-ordering
  - Selective Acknowledgments
  - Window Scaling
  - Explicit Congestion Notification
  - Forward RTO recovery and RTO response
  - Early-Retransmit

Policy-Based TCP Optimization

Policy-based TCP optimization enables you to create and save TCP optimization profiles and have them applied to network traffic based on policy input, such as Policy Manager conditions or PCRF or RADIUS attribute values.

A TCP optimization profile is a set of TCP parameter settings created specifically to optimize TCP for a particular operator network or for certain conditions. For example, you might have a profile designed to optimize TCP for a 3G network. This profile would be applied through a NetScaler policy when one or more NetScaler policy expressions are matched, such as Network Type = 3G

TCP optimization profiles are applied on a per-connection (TCP socket) basis. If a network condition changes, the TCP optimization profile being applied to the connections can also be changed. A simple example would be the case where subscribers on different types of networks (like 3G and LTE) are being handled by the same Traffic Manager platform. If the network type specifies 3G, as indicated by RADIUS or PCRF, this policy input can be used to set the TCP optimization parameters accordingly (in this case, by using the parameters defined in a 3G TCP optimization profile). If the network type for a connection is LTE, the TCP optimization can use an LTE profile. This happens automatically based on the Policy Manager configuration. No operator intervention is required.

Some other usage cases might be to set a particular TCP optimization profile to be used during a certain time of day (for example, during busy hours), or based on a range of network IP addresses.

TCP optimization profiles are applied to client-side TCP connections at the beginning of a TCP connection. The parameters specified in the TCP optimization profile last until the connection is terminated.

TCP Profiles, Global TCP Parameters, and Precedence

Not every TCP parameter recommended for LTE/3G optimization can be configured by using policy-based TCP profiles. For example, TCP ABC, F-RTO, and Slow Start After Idle can only be changed globally (see section 5.1.1). Therefore, even if you are using policy-based TCP profiles, some additional TCP parameters may need to be tuned globally for optimal network performance.

If a TCP optimization profile is being used in an activated policy, the TCP optimization parameters contained in the profile take precedence over the global TCP settings. If the TCP optimization profile is not being used by an activated policy, it will not be in effect and the global TCP settings will be used. Because TCP profiles do not allow configuration of all the parameters that can be configured globally, an activated TCP profile will only take precedence over those global settings that exist in both the global and TCP profile configuration sets. For parameters that do not exist in the TCP profile, the global values are used.

[3] Additional congestion control options can be configured by Support users.

TCP Nile Congestion Control

TCP Nile is a Citrix-proprietary, optimized TCP congestion control that is specifically designed to provide higher throughput and more effective utilization of wireless network links (as compared to traditional ones, such as Westwood+, BIC, or Illinois).

In general, TCP congestion handlers are grouped into two main categories: loss based and delay based. Loss based congestion handlers use the packet loss information as a signal for congestion, whereas delay based congestion handlers continuously measure the round trip time and make congestion decisions based on variations in round-trip time measurements.

TCP Nile is classified in a class of congestion handlers that use both loss and delay as input for congestion window adjustment, frequently referred to as hybrid congestion handlers.

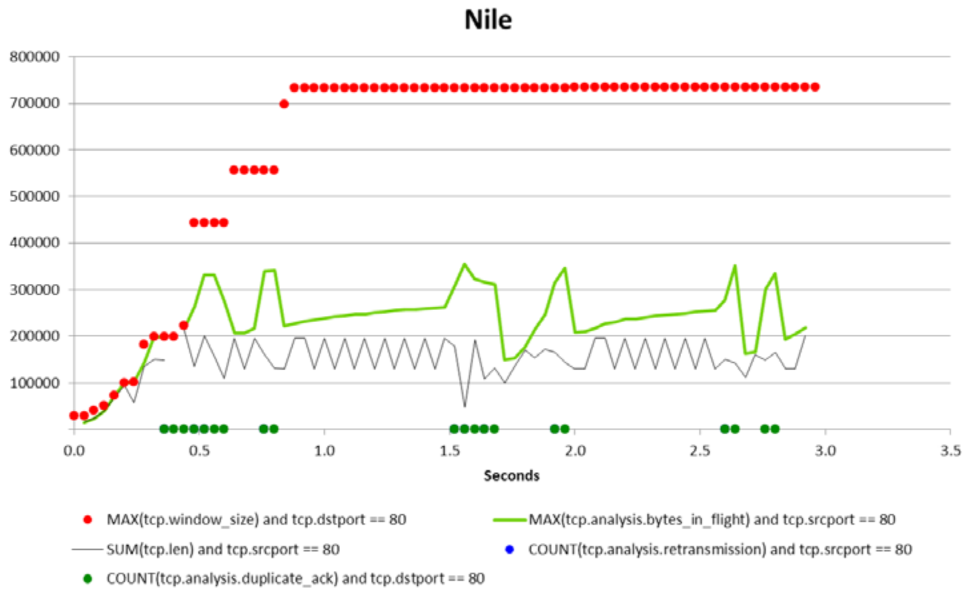
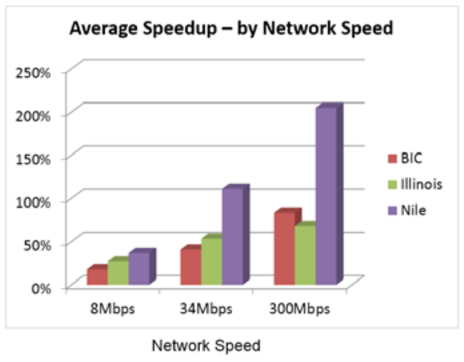


Figure 5: TCP Nile downloading 10MB file (example)

TCP Nile implements additional key improvements that lead to increased performance:

•Improves handling in conditions increased jitter, random loss and retransmission
•Makes TCP congestion window adjustment more agile to fluctuating bandwidth
•Performs better and more accurate delay measurement and congestion detection

The results are based on comprehensive testing (2700 test cases) with different network settings (link speed, RTT, latency jitter, packet loss rate, queue size and session duration)



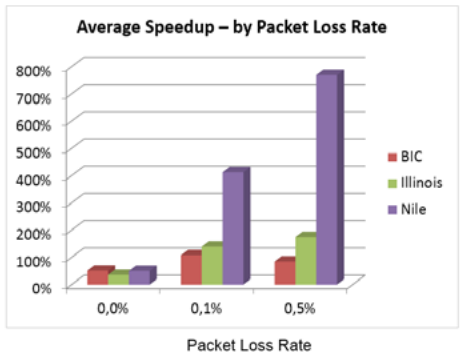


Figure 6: Citrix lab results comparing BIC, Illinois and Nile

Conclusion

Mobile networks have unique characteristics that may impact TCP performance. By leveraging NetScaler's TCP optimization solution, mobile operators can speed up file download, reduce webpage load time and decrease video startup time. The end result is higher QoE and increased levels of satisfaction for the subscribers as shown in figure 6.

References
http://www.mclab.info/MobileAccelerator_IWCMC2011.pdf