WHITE PAPER

Optimizing TCP: Nagle's Algorithm and Beyond

Tips for using Nagle's Algorithm, TCP_QUICKACK, Delayed ACK, and other TCP optimization settings that are tough to understand, but crucial to get right

Abstract

The Transmission Control Protocol/Internet Protocol stack (TCP/IP) is foundational to today's digital enterprises, and its quality of performance has repercussions that everyone in IT should care about—not just network engineers. TCP is where the network and application meet, and is often ignored by network engineers and application teams alike when troubleshooting performance issues.

At ExtraHop, we've learned through our work with hundreds of enterprise customers that many network and application performance problems are the result of a poorly tuned TCP/IP implementation. Understanding how to diagnose TCP issues, and how to fix them, is a great asset for any IT professional.

Office workers accessing internal applications. Shoppers browsing a website. Doctors logging into a workstation. All of these people will have a worse experience if network, application and operations teams aren't optimizing TCP configurations for their particular use case.

After reading this paper, you'll have a better understanding of some oft-misunderstood methods for improving TCP performance, which will make it easier to troubleshoot network performance problems in the future, and will give you a leg-up when it comes to fine-tuning your network.

Table of Contents

A Portrait of TCP as a Young Protocol

TCP was introduced in December, 1974, (RFC 675) with the explicit purpose of providing "a reliable process-to-process communication service in a multinetwork environment." The protocol was developed by fathers of the Internet Vint Cerf and Bob Kahn, and introduced in the seminal paper *A Protocol for Packet Network Intercommunication*.[1] The protocol continued to be refined, but the core principles were laid out in that first paper and a subsequent RFC published in 1981 (RFC 793).

The networks of 1981 were much smaller, had fewer nodes, fewer processes operating, and far lower-bandwidth connections for intercommunication than today's networks. A 1981 TV news segment estimated that there were "two-to-three thousand home computer owners" in the San Francisco Bay area (lol), around 500 of whom had used the internet to access the brand new electronic editions of the city's local newspapers.[2]

Even though global interoperability between networks was the explicit goal of TCP, its creators couldn't really anticipate the eventual enormity of the global multi-network environment that their protocol would come to enable.

Optimizing TCP

As packet-switched networks became part of the foundational infrastructure of every business, and the public internet scaled from thousands of users to billions within a few decades, TCP marched on as the core transport protocol that allowed various systems to talk to each other. TCP was so crucial to networks that it became impractical to use anything else in its place.

In fact, TCP was partly created to solve reliability and scaling issues with existing protocols. With that in mind, some of the original core principles of TCP were:

- Accuracy and confirmation of transmission - The packets sent should be identical to the packets received, and the receiver should let you know the data got there.
- Reliability - The packets sent should all arrive, and should be in the correct order.
- Flow control - Packet flow should adapt to the capacity of the network and endpoints. TCP manages flow between endpoints, and between the IP layer below TCP and other layers above it.

These principles make TCP a phenomenal transport protocol for many use cases involving networked communications, but as the complexity and dynamism of network communications have increased, many scenarios have arisen where TCP must be used, but its core principles are not actually optimized for the situation.

The next section will discuss commonly misunderstood and misused settings and algorithms used for fine tuning and optimizing TCP, including how to identify situations where the settings are being used incorrectly, or aren't being used at all when they should be.

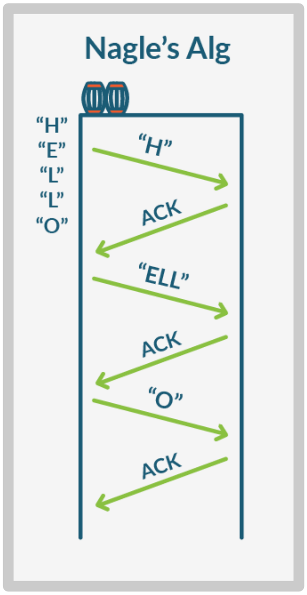
[1] https://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/cerf74.pdf
[2] https://www.youtube.com/watch?v=5WCTn4FljUQ

The Settings and Algorithms

Nagle's Algorithm & TCP_NODELAY

What is Nagle's Algorithm?

Nagle's Algorithm is a setting that improves the efficiency of network communications by reducing the total number of TCP packets that need to be transmitted. Enabling Nagle's algorithm causes TCP implementations to suppress their normal behavior, that of sending packets immediately when data is received from an application, in cases where packets might be sent with a very small amount of payload. For example, if an application emits a single byte of data and tells TCP to send it, Nagle's Algorithm will force the system to wait until more data arrives to fill up the packet. Since a TCP packet includes 40 bytes of header information, sending a packet with a single byte of payload, also called a tinygram, is analogous to moving your residence by moving each piece of furniture on its own truck rather than efficiently packing a single truck.



Nagle's Algorithm was introduced to TCP in the mid 1980s by John Nagle (RFC 896) to confront the growing issue of network congestion resulting from excess tinygrams.

It became necessary to let the owner of a network or application choose how TCP handled their traffic. They could either:

1. Enable Nagle's Algorithm (the default setting for many TCP implementations), causing some bits to be delayed in favor of packing them more efficiently into TCP packets and improving the header-to-payload ratio.
2. Disable Nagle's Algorithm, allowing packets with only a single byte of payload to be transmitted instantly, to allow tiny bits of info to be transmitted as quickly as possible, even if it results in inefficient network utilization.

Should You Disable Nagle's Algorithm using TCP_NODELAY?

Maybe. Here's how to find out:

The answer is different for each network, and it largely depends on your traffic mix, whether or not you're already seeing network congestion or excess tinygrams, and what type of behavior your network needs to exhibit to deliver the best experience.

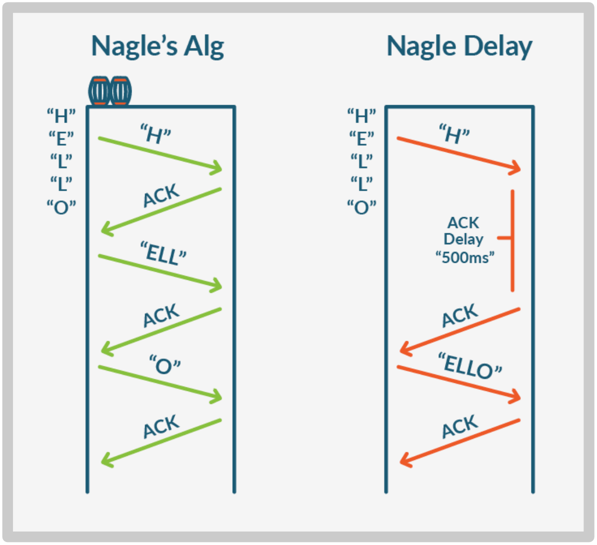
Answering this question requires a process of trial and error and fine-tuning for your specific conditions, but the steps listed below should be a good start in most situations. These require having some kind of monitoring system that can tell you how many Nagle delays or tinygrams are occurring on your network, or at least give you a general metric for network congestion in your environment. We'll use ExtraHop in this example, but as long as your monitoring can report Nagle delays, tinygrams, and other network-congestion metrics, you should be able to use this process.

1. Set a Baseline: Look at the TCP stats for whichever server, load balancer, or other endpoint you suspect may be experiencing or causing latency due to Nagle's algorithm. Check this stat at a few different times over a period of a few days to get a general idea of what's "normal" for that device. You don't need to get too scientific at this stage, just get a feel for what's normal.

2. Change Something: Once you have a baseline, it's time to start tweaking. Toggle Nagle's algorithm on or off for a few endpoints, depending on what state you started with.
3. Check your status: Check in on your Nagle delays and tinygrams over the next few days. Are you seeing more or fewer? Are there other performance changes in the part of the network you're looking at?
4. Soak, Rinse, Repeat: Whether you turned Nagle's algorithm on or off, switch it back to how it was before, and do the same informal data gathering process. This isn't rocket science, you're just making tweaks in a somewhat controlled environment, but you can probably get enough insight to figure out whether Nagle is right for this particular switch or router. Then you can apply the process to other areas of the network where you suspect it could make a difference.

There are no universal rules around this, but generally, applications that require immediate transmission for good user experience will not benefit from Nagle's algorithm. This is especially applicable to remote and virtual desktops (Citrix ICA, particularly) and other applications that require immediate delivery of data and rapid feedback from users to function as intended.

Conversely, applications that usually move large files around, such as storage protocols, don't typically see much improvement with Nagle's algorithm enabled. They're already filling packets as full as they can go, rather than sending packets instantly when any amount of data arrives.



What Is TCP_NODELAY, and Should You Use It?

TCP_NODELAY is a socket setting that disables Nagle's algorithm and allows tinygrams to be sent across the network. If you examine the factors listed above and determine that Nagle's algorithm is hurting the performance of your network, setting TCP_NODELAY is one way to address the problem. This is rarely a good option, since it allows network-congesting tinygrams to flow freely, but you may encounter circumstances where you're glad this setting is available.

TCP_QUICKACK and Delayed ACK

What Is TCP_QUICKACK?

TCP_QUICKACK is a setting that allows TCP endpoints to acknowledge the receipt of data instantly in situations where they would normally wait to see if more data would be arriving.

TCP/IP has a built-in mechanism for endpoints to acknowledge that they have received data, so that the sender knows the message was received. This acknowledgement is called an ACK segment, and in a typical TCP/IP implementation with Nagle's algorithm enabled, a sender will await the ACK segment before sending further data. That effectively means Nagle's algorithm only allows a single TCP packet to pass across any given connection at once, which can introduce a huge amount of latency.

Setting TCP_QUICKACK in this scenario allows the receiver to send an ACK immediately in situations where it would normally wait to see if any further data was going to arrive over the connection first.

Should You Disable Delayed ACKs Using TCP_QUICKACK?

Almost always.

TCP_QUICKACK offers some of the benefits, and none of the downsides, of TCP_NODELAY. John Nagle himself, the creator of Nagle's algorithm, had this to say on a comment thread comparing TCP_NODELAY to TCP_QUICKACK:

"Set TCP_QUICKACK. If you find a case where that makes things worse, let me know."

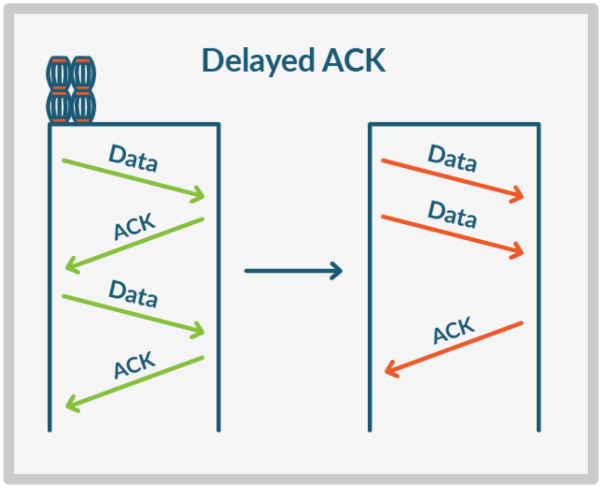
What Is Delayed ACK?

Delayed ACK, or delayed acknowledgement, is a default behavior of TCP that combines several ACK segments into a single response under some circumstances to reduce network overhead. Instead of sending an ACK immediately upon receiving data, Delayed ACK allows TCP to combine the ACK with the window adjustment and even the response data, somewhat reducing network overhead.

Should You Leave Delayed ACK Enabled?

Probably not.

Delayed ACK interacts terribly with Nagle's algorithm, which is turned on by default for many systems. This means enabling Delayed ACK might have unforeseen consequences that would be difficult to diagnose or pinpoint (especially if you have a heterogeneous traffic mix across multiple switches, routers, and VLANs).



Beyond that, Delayed ACK just isn't that efficient. At best, you're getting a threefold reduction in network overhead. Using Nagle's algorithm by itself can yield 40x improvement in header to payload ratio, but turning on delayed ACK and Nagle's algorithm at the same time negates the value of both.

See How Deep the TCP Rabbit-Hole Goes …

This paper has covered some of the TCP issues that have proven most difficult to detect, and most impactful when fixed, for actual ExtraHop customers. TCP is integral to every network, and is complex enough that any issues with it will be tough to diagnose for anyone without direct previous experience with the exact issue at hand. It would be hard to outline every possible issue without writing an entire textbook. (In fact, if that's the approach you're looking for, we'd recommend TCP/IP Illustrated, Volume 1: The Protocols.[3])

However, in the interest of helping you prepare as much as possible to recognize issues that may be rooted in TCP, here's a list of other terms to know, optimization algorithms or concepts you may encounter, and links to resources to learn more about them if they sound like they might be relevant to the issues you have right now.

- Retransmission Timeout (RTO): This is what happens when a TCP sender doesn't receive acknowledgement, and stops sending momentarily to give the recipient time to catch up. This can introduce a lag of over 1 second per transaction. Multiplied across thousands or millions of transactions, this can create enormous amounts of latency and app slowdowns that are tough to diagnose. Learn more about RTOs: http://xtra.li/TCPRTOs
- Silly Window Syndrome: This is a situation that arises when a TCP sender and receiver have mismatched expectations of each other. A TCP sender may send data too fast, and the receiver may not be able to process and acknowledge the data fast enough, causing it to advertise a smaller and smaller amount of "usable window" for each transmission, until the amount is so small it's silly. Learn more about silly window syndrome: http://xtra.li/SillyWin
- Tinygrams: A tinygram is a TCP packet with less payload than header. Tinygrams can result from various timing issues and misconfigurations in a TCP implementation. Learn more about Tinygrams: http://xtra.li/tnygrm
- Karn's Algorithm: Karn's algorithm is a setting that makes TCP more effective at estimating the round-trip time of transmissions. It is almost always turned on by default, and almost always beneficial, but there are situations where it isn't, and it pays to understand these edge cases. Learn more about Karn's Algorithm: http://xtra.li/KarnsAlgo
- TCP PAWS Dropped Syns: Protection Against Wrapped Sequence numbers (PAWS) was added to TCP in the early nineties as internet speeds were rapidly increasing, to protect against older duplicate messages corrupting still-open connections. PAWS uses timestamps to determine whether packets are duplicates, so any inconsistency between clocks of senders and receivers (more common in situations where messages are sent between remote datacenters) can cause confusion and result in dropped SYNs. At high enough volume this can seriously impact network performance. Learn more about PAWS dropped SYNs: http://xtra.li/PAWS-SYN

[3] http://www.kohala.com/start/tcpipiv1.html

Conclusion: Always Be Checking TCP

The ultimate lesson here is that a TCP implementation that's misaligned with the needs of the apps or networks it supports can be the source of a lot of strife. Problems arising with TCP can be tough to diagnose, and are often blamed on other causes, or even used as justification to buy more networking hardware and bandwidth. By keeping an eye on your TCP situation, you stand to save yourself and your company time and money, and likely help your peers on the network or application team get more sleep when an issue arises.

In addition to its industry-leading application protocol analysis, the ExtraHop platform also provides incredible insight into TCP by reporting on Nagle delays, tinygrams, retransmission timeouts, and other metrics that can be strong indicators of network latency being introduced by TCP misconfiguration.

Try the ExtraHop interactive online demo to see exactly how easy it can be to diagnose TCP issues and discover tons of other optimization opportunities in your IT environment: www.extrahop.com/demo

Read how Axstores, a leading retailer in the Nordic region, solved TCP misconfigurations in their new warehouse management system before rolling it out into production: https://www.extrahop.com/platform/stories/axstores/

"ExtraHop does more than just application monitoring—it gives us visibility into core network services that are crucial when optimizing performance."

Stefan Pörn, IT Operations Manager, Axstores

ExtraHop Networks, Inc.
520 Pike Street, Suite 1700
Seattle, WA 98101 USA

www.extrahop.com
info@extrahop.com
T 877-333-9872
F 206-274-6393

Customer Support support@extrahop.com
877-333-9872 (US)
+44 (0)845 5199150 (EMEA)