Internet Engineering Task Force
INTERNET-DRAFT
draft-minshall-nagle-00

Greg Minshall
Siara Systems
December 18, 1998

A Suggested Modification to Nagle's Algorithm

Status of This Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ``work in progress.''

To view the entire list of current Internet-Drafts, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

This draft proposes a modification to Nagle's algorithm (as specified in RFC896) to allow TCP, under certain conditions, to send a small sized packet immediately after one or more maximum segment sized packet.

Abstract

The Nagle algorithm is one of the primary mechanisms which protects the internet from poorly designed and/or implemented applications. However, for a certain class of applications (notably, request-response protocols) the Nagle algorithm interacts poorly with delayed acknowledgements to give these applications poorer performance.

This draft is NOT suggesting that these applications should disable the Nagle algorithm.

This draft suggests a fairly small and simple modification to the Nagle algorithm to preserve Nagle as a means of protecting the internet while at the same time giving better performance to a wider class of applications.

Introduction to the Nagle algorithm

The Nagle algorithm [RFC896] protects the internet from applications (most notably Telnet, at the time the algorithm was developed) which tend to dribble small amounts of data to TCP. Without the Nagle algorithm, TCP would transmit a packet, with a small amount of data, in response to each of the application's writes to TCP. With the Nagle algorithm, a first small packet will be transmitted, then subsequent writes from the application will be buffered at the sending TCP until either i) enough application data has accumulated to enable TCP to transmit a maximum sized packet, or ii) the initial small packet is acknowledged by the receiving TCP. This limits the number of small packets to one per round trip time.

While the current Nagle algorithm does a very good job of protecting the internet from such applications, there are other applications, such as request-response protocols (with HTTP 1.1 being a topical example) in which the current Nagle algorithm produces non-optimal results. In this context, the Nagle algorithm is interacting with TCP's ``delayed ACK'' policy [RFC1122].

Delayed ACKs

A receiving TCP tries to avoid acknowledging every received data packet. This process, known as ``delayed ACKing'' [RFC1122], typically causes an ACK to be generated for every other received (full-sized) data packet. In the case of an ``isolated'' TCP packet (i.e., where a second TCP packet is not going to arrive anytime soon), the delayed ACK policy causes an acknowledgement for the data in the isolated packet to be sent within 200 milliseconds of the receipt of the isolated packet. (The way delayed ACKs are implemented in some systems causes the delayed ACK to be generated anytime between 0 and 200ms; in this case, the average amount of time before the delayed ACK is generated is 100ms.)

The interaction of delayed ACKs and Nagle

If a TCP has more application data to transmit than will fit in one packet, but less than two full-sized packets' worth of data, it will transmit the first packet. As a result of Nagle, it will not transmit the second packet until the first packet has been acknowledged. On the other hand, the receiving TCP will delay acknowledging the first packet until either i) a second packet arrives (which, in this case, won't arrive), or ii) approximately 100ms (and a maximum of 200ms) has elapsed.

When the sending TCP receives the delayed ACK, it can then transmit its second packet.

In a request-response protocol, this second packet will complete either a request or a response, which then enables a succeeding response or request.

Note two (related) bad results of the interaction of delayed ACKs and the Nagle algorithm in this case: the request-response time may be increased by up to 400ms (if both the request and the response are delayed); and, the number of transactions per second is substantially reduced.

A proposed modification to the Nagle algorithm

The current Nagle algorithm can be described as follows:

> If a TCP has less than a full-sized packet to transmit, and if any previous packet has not yet been acknowledged, do not transmit a packet.

The proposed Nagle algorithm modifies this as follows:

> If a TCP has less than a full-sized packet to transmit, and if any previous less than full-sized packet has not yet been acknowledged, do not transmit a packet.

In other words, when running Nagle, only look at the recent transmission (and acknowledgement) of small packets (rather than all packets, as in the current Nagle).

(In writing the above, I am aware that TCP acknowledges BYTES, not packets. However, expressing the algorithm in terms of packets seems to make the explanation a bit clearer.)

Implementation of the modified Nagle algorithm in a system

The current Nagle algorithm does not require any more state to be kept by TCP on a system. SND_NXT is a TCP variable which names the next byte of data to be transmitted. SND_UNA is a TCP variable which names the next byte of data to be acknowledged. If SND_NXT equals SND_UNA, then all previous packets have been acknowledged.

The proposed modification to the Nagle algorithm does, unfortunately, require one new state variable to be kept by TCP. SND_SML is a TCP variable which names the last byte of data in the most recently transmitted small packet.

An implementation could be as follows:

> 1. When transmitting a small packet, record the sequence number of the last byte of the small packet in SND_SML.

2. When deciding whether or not to transmit a small packet, check to ensure that SND_SML is less than, or equal to, SND_UNA.

A Failure Mode

If an application sends a large amount of data, followed by a small amount of data, followed by a large amount of data, the current Nagle algorithm would perform better than the proposed modification. The current Nagle algorithm would send at most one small packet (possibly the last packet), delaying the middle (small) amount of data which would allow the application to send the following large amount of data; the proposed Nagle algorithm would send two small packets (the middle packet, plus possibly a last packet).

A separate, but desirable, system facility

In addition to the Nagle algorithm (or the modification proposed by this draft), it would be desirable for a system providing TCP service to applications to allow the application to set TCP into a mode in which the TCP would only transmit small packets at the explicit direction of the application. For example, a system based on BSD might implement a socket option (using setsockopt(2)) SO_EXPLICITPUSH, as well as a flag to sendto(2) (possibly overloading the semantics of an existing flag, such as MSG_EOF).

In this scenario, an application would set a socket into SO_EXPLICITPUSH mode, then enter a mode of writing data to the socket and, at the last write, using send(2) with the MSG_EOF flag. The underlying TCP would recognize the MSG_EOF flag as an indicator to transmit the (possibly) small packet.

Like the proposed modification to the Nagle algorithm, this is fairly simple to implement.

If a system were to implement this interface, it would be important to NOT disable Nagle when using this interface. In other words, when using this interface, the default mode for TCP would be to NOT transmit a small packet (even in the presence of MSG_EOF) if a previously transmitted small packet was as yet unacknowledged.

Note, also, that implementing this interface does not eliminate the desirability of using the modification of the Nagle as the default for applications. More sophisticated networking applications might well use the new interface, but naive applications will often be adequately served by the modified Nagle algorithm.

Acknowledgements

Jim Gettys, Henrik Frystyk Nielsen, Jeff Mogul, and Yasushi Saito, as well as a message forwarded to the end2end-interest list by Sean Doran, have motivated my current interest in the Nagle algorithm. John Heidemann's work related to the Nagle algorithm has informed some of the thinking in this draft; discussions with John have also been helpful. Members of the End-to-End Research Group (under the direction of Bob Braden) patiently listened to my discussion of the current state of the Nagle algorithm and to the modifications proposed in this document.

Security Considerations

The Nagle algorithm does not have major security consequences.

Implementation of this algorithm should not negatively impact the performance of the internet. The negative impact of implementation of this algorithm should be significantly less than disabling the Nagle algorithm.

References

[RFC896] Nagle, J., "Congestion control in IP/TCP internetworks", Jan-06-1984.

[RFC1122] Braden, R. T., "Requirements for Internet hosts - communication layers", Oct-01-1989.

Author's Addresses

Greg Minshall
Siara Systems
1399 Charleston Road
Mountain View, CA 94043
USA

<minshall@siara.com>