TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Informational
Expires: April 19, 2008

M. Jethanandani
Cisco Systems
M. Bashyam
Ocarina Systems, Inc

October 17, 2007

TCP Robustness in Persist Condition
draft-mahesh-persist-timeout-02

Status of this Memo

Copyright Notice

Abstract

This document describes how a connection can remain infinitely in persist condition, and its Denial of Service (DoS) implication on the system, if there is no mechanism to recover from this anomaly.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Table of Contents

1. Introduction

RFC 1122 [RFC1122] Section 4.2.2.17, page 92 says that: A TCP MAY keep its offered receive window closed indefinitely. As long as the receiving TCP continues to send acknowledgments in response to the probe segments, the sending TCP MUST allow the connection to stay open.

The RFC goes on to say that it is important to remember that ACK (acknowledgement) segments that contain no data are not reliably transmitted by TCP. Therefore zero window probing SHOULD be supported to prevent a connection from hanging forever if ACK segments that re-opens the window is lost.

While the RFC is clear why the sender needs to continue to probe the receiver, it is not clear why this process needs to be indefinite, particularly if the receiver continually responds with a ACK and a window of zero. This draft documents a negative consequence of this indefinite attempt by the sender to probe for the receiver's offered window.

One negative consequence of this indefinite attempt is that it makes the sender vulnerable to a connection and send buffer exhaustion attack by one or more malicious receivers. This leads to a Denial of Service (DoS) where legitimate connections stop getting established and well behaved already established connections stop making progress in terms of data transmission.

Having the sender accumulate buffers and connection table entries when the receiver has deliberately and maliciously closed the window can ultimately lead to resource exhaustion on the sender. This particular dependence on the receiver to open its zero window can be easily exploited by a malicious receiver to launch a DoS attack against the sender.

The condition where the sender has at least one buffer in the send queue is referred to as persist condition. In this condition the sender is waiting indefinitely for the receiver to open up its window.

Resources that are compromised due to this sender behavior include connections and send buffers, since both of these are finite pools in any server.

The problem is applicable to TCP and TCP derived transport protocol like SCTP.

We have done some experimention to demonstrate this problem and

looked at how many servers on the Internet are susceptible to it. The rest of the draft will detail the experiment, suggest how the problem needs to be addressed, why we believe it is the right solution and what role application can play in solving this problem.

For TCP to persist indefinitely makes the end point vulnerable to a DoS attack. We therefore clarify the purpose of zero window as described in RFC 1122 and suggest that TCP end point SHOULD NOT keep a connection in persist condition for an indefinite amount of time.

In most implementations, TCP runs in kernel mode as part of the operating system. In this mode the operating system may share the same address space as TCP. For the purposes of discussion, this draft considers TCP protocol implementation to be a separate module responsible for all resources such as buffers and connection control blocks that it borrows from the operating system. The operating system can enforce the maximum number of buffers it is willing to give to TCP but beyond that it lets TCP decide how to manage them.

2. Denial of Service Experimentation

The effect of the receiver that stops reading data is that the sender continues to send data till the receiver advertised window goes to zero at which time the connection enters persist condition. Since the sender has more buffers with data for the client, it will continue to probe the receiver. If the sender is servicing several such clients the effect compounds itself to the extent that the sender runs out of buffers and/or connection resources. The sender at this point cannot service new legitimate connections and even the existing connections start seeing degraded service. Further, each connection reserves a connection control block, which are of a finite amount. Several connections in persist condition can exhaust the connection control block pool.

To demonstrate the problem we wrote a user level program that puts TCP connections on the HTTP server in persist condition. The client can run on any machine and does not require a change in the kernel or the operating system.

The client opens a TCP connection to the HTTP server with a advertised MSS of 1460. It then sends a GET request for a large page. The page size is large enough to ensure that the connections send buffer always has more data than receivers maximum advertised window. Once the window has been opened, the client application stops reading data resulting in TCP closing the window and advertising zero window towards the sender. For each request of a multi-megabyte response, the connection can result in the sender

holding on to all the requested data minus the receivers advertised window, in its send queue. If the receiver never closes the connection, the server will continue to hold that data indefinitely in its send queue.

The same program was then run from each client with it opening one thousand connections towards the HTTP server. This was run from several different machines with the result that now the server was holding onto several thousand connections, each with more than one megabyte worth of data on the send queue.

After verifying this behavior in the laboratory against both a Apache and a IIS server, we then proceeded to test HTTP servers on the Internet. To verify this behavior we needed to open only few connections towards the servers. We chose three well known sites, identified here as Site A, Site B and Site C for our test. We then ran a network analyzer on the client machine to monitor the behavior of the connection. These were our observations.

Connections to Site A went into ESTABLISHED state and after receiving receivers advertised window worth of data went into persist condition. The connection persisted in this mode for approximately 11 minutes and was then RST by the server.

Connections to Site B went and stayed in ESTABLISHED state. They stayed in that state as long as the client kept the connection open. The server in this case was Apache version 2.0. The size of the file requested was 12.12M. The client received 200K worth of data and the rest of the data was either queued on the send queue or in application.

Connection to Site C went into and stayed in ESTABLISHED state. They too stayed in that state as long as the client kept the connection open, which was as long as five days. The server in this case was a IIS server version 6.0. The size of the requested page was 1.09M (a pdf file). The client had received 200K worth of data and the rest of the data was either queued on the send queue or in application.

As can be seen from the experimentation the behavior of TCP varied greatly between different sites. Site A appears to implement a User Time Out (UTO) or application timeout on their connections. That allowed it to clear the connections. However, once it was known what the fixed timeout was, it was easy to modify the client program to open another set of connections after the timeout. We discuss the role of application and the use of UTO in a later section. It was difficult to establish how much data was sitting on the send queue of each one of these public servers as that depends on send socket buffer size and how much data was written by the application.

Please note that it is not required for the client to issue a request for a large page or for the server to open its window completely to reproduce the DoS scenario. A page size larger than the advertised window size is enough. We decided to do it with a larger response because it enabled us to reproduce the problem with fewer number of connections and client machines.

Persist condition clearly has a more significant impact on servers that deal with a large number of connections (e.g. 200-300K connections), than on end workstations that might deal with a few connections at a time. This is because the server has a finite number of buffers for a larger pool of connections. With dynamic allocation of buffers, each connection is given resources as it needs them. A high water mark set on each connection prevents the number of enqueued buffers exceeding that mark till such time that the number of buffers fall below a low water mark. However, that in itself does not solve the problem as the high water mark is more than the advertised window size.

3. Solution

The current behavior of the connection in persist condition SHALL continue to exist as the default behavior. The solution proposed will control the amount of time a TCP sender will spend in persist condition waiting for receiver to open its window. Outlined are some of the ways that this can be achieved. Default values are suggested values and the implementor is free to choose their own value.

If the administrator of the system decides to use the proposed solution, they will need to enable it explicitly. Optionally, the administrator can configure a minimum and maximum threshold values for connections and buffer resources for the total pool. Default values of 60 and 80% of the total pool for minimum and maximum respectively are assumed.

While implementing the solution it is important to make sure that legitimate and well behaved receivers are not penalized for offering zero or reduced window. Hence the solution needs to be robust. It is also important that the solution be adaptive. While resources are plenty, connections are allowed to spend more time in persist condition. However, as resources become scarce the connections are aborted sooner.

A fixed timeout value is not a effective solution. Malicious clients can discover the timeout value and can (re)launch an attack after the fixed timeout period.

If the solution is enabled, the global persist-condition-expiry -time value will be set to infinity (or a very large value). Thereafter it will adapted based on system resources availability. The persist-condition-expiry-time is bounded above by the default value of 60 seconds and a minimum value of five seconds (or minimum persist timeout). The administrator has the option to change the default value. To prevent wild fluctuations in this timeout value, the time will be recomputed only when resources change by at least 1%. If the total pool of resources is less than minimum threshold, the persist-condition-expiry-time value is set to infinity (a very large value). If the resource utilization increases to being between minimum and maximum, then persist-condition-expirty-time is first set to the default value and thereafter decreased additively by two seconds. If resources exceed the maximum, the persist-condition-expiry-time is decreased multiplicatively by a factor or two. If the resource utilization starts to decrease then persist-condition-expirty-time is increased additively by four seconds. If the utilization falls below minimum, the time is set to infinity.

The solution focuses on figuring on how to keep track of connections in persist condition. The configured option of persist-condition-expiry-time implies how long the connection will be allowed to stay in persist condition. When the connection enters persist condition, i.e. the receiver advertises a window of zero, the value of current time - now, is saved in the connection entry. This entry is called persist-condition-entry-time. In addition, the sequence number on the connection is stored as persist-condition-sequence-number. Thereafter every time the persist timer expires or when an ACK is received that continues to advertise zero window, a check is done to make sure that the difference between current time and persist-condition-entry-time is not more than persist-condition-expiry-time. If it is then the connection is aborted and the connection resources are reclaimed.

The receiver's silly window avoidance mechanism will make sure that the receiver cannot read a small amount of data and fool the sender into taking it out of persist condition.

For the solution to be robust, it is also important to determine which connection among the set of connections in persist condition is selected to be terminated. To implement this effectively, we maintain two priority queues of connections in persist condition, one based on the amount of data in the send queue and another based on the persist-condition-entry-time, i.e. when the connection entered persist condition.

Whenever a buffer resource is required and the resource utilization is more than the maximum, the connection with the highest amount of

data in the send queue is dropped, and its buffers recycled. Whenever a connection resource is required and the connection utilization is higher than the maximum, the connection with the oldest persist-condition-entry-time is selected and dropped. This achieves fairness by penalizing the connection which are consuming the most resources.

4. Role of Application

Applications are agnostic to why TCP connections are not making progress in terms of data transmission. TCP connections may not be able to transmit data for a variety of reasons. Today TCP does not provide an indication of the progress of the connection explicitly. It is up to the application to conclude based on an examination of the send queue backlog or implement a UTO as defined in RFC 793 [RFC0793]. A lot of commonly used applications do not implement the UTO scheme, e.g. World Wide Web (WWW). Even if the application did implement a UTO scheme, all applications running the system need to have implemented the UTO for the solution to be effective. A single application that has not implemented the UTO can cause the entire system to be impacted negatively.

There are cases where the system is application agnostic. A classic case of this is a TCP proxy. In that particular case, there is no end application that can be informed of the state of the connection for the application to take action.

Resources like TCP buffers are system wide resources and are not tied to any particular application. TCP needs to be able to monitor resource usage system wide when connections are in persist condition. The application does not have the connection's sender state knowledge to implement a robust and adaptive solution such as the one outlined here.

Applications can assist TCP's role in solving this problem. They can register for an event notification when the TCP connection enters or exits persist condition. They can use the notification mechanism to implement their own scheme of deciding which persist connections to clear. They can also suggest timeout or retry values to TCP.

5. IANA Considerations

This document makes no request of IANA.

6. Security Considerations

This document discusses one security consideration. That is the possible DoS attacks discussed in Section 2.

7. Acknowledgements

Thanks to Anantha Ramaiah who spent countless hours reviewing, commenting and proposing changes to the draft. Ted Faber helped us in clarifying the objective of this RFC. Thanks also to Fred Baker and Elliot Lear for providing their feedback on the draft.

Our thanks to Nanda Bhajana who helped arrange the test setup to be able to reproduce the DoS scenario.

8. References

8.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

Appendix A. An Appendix

Authors' Addresses

Mahesh Jethanandani
Cisco Systems
170 West Tasman Drive
San Jose, California 95134
USA

Phone: +1-408-527-8230
Fax: +1-408-527-0147
Email: mahesh@cisco.com
URI: www.cisco.com

Murali Bashyam
Ocarina Systems, Inc
Fremont, CA
USA

Phone:
Fax:
Email: mbashyam@ocarinatech.com
URI:

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).