ESXi NFS Read Performance: TCP Interaction between Slow Start and Delayed Acknowledgement

Performance Case Study / ESXi 7.0 - May 11, 2020

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com

Table of Contents

Executive Summary

VMware performance engineers observed, under certain conditions, that ESXi IO (in versions 6.x – 7.0) with some NFS servers experienced unexpectedly low read throughput in the presence of extremely low packet loss, due to an undesirable TCP interaction between the ESXi host and the NFS server. In this paper, we explain how this TCP interaction leads to poor ESXi NFS read performance, describe ways to determine whether this interaction is occurring in an environment, and present a workaround for ESXi 7.0 that could improve performance significantly when this interaction is detected. In our experiments with ESXi NFS read traffic from an NFS datastore, a seemingly minor 0.02% packet loss resulted in an unexpected 35% decrease in NFS read throughput.

A key lesson of this paper is that seemingly minor packet loss rates could have an outsized impact on the overall performance of ESXi networked storage. If you are using ESXi networked storage and have workloads that are highly performance sensitive, you should consider taking steps to identify and mitigate these undesirable interactions if appropriate.

This TCP interaction has also been observed for ESXi SWiSCSI. See KB 1002598 for details [1].

Introduction

In this paper, we explain how a very minor packet loss rate (as low as 0.02%) on the network could trigger an undesirable interaction between TCP slow start on a sender and delayed acknowledgement on the receiver, leading to unexpectedly poor data transfer throughput. We then describe a scenario in which this interaction manifests itself in the context of ESXi read operations from an NFS server.

To determine if this undesirable TCP interaction is present in a given environment, we capture and analyze the network traffic between a VMware ESXi™ host and an NFS server. We demonstrate how key signatures of this interaction can be identified using Wireshark analysis. The undesirable interaction can be suppressed by disabling TCP delayed acknowledgements (delayed ACKs) on the NFS client of the ESXi host. In our experiments, we observed significant throughput improvements for large IO-sized read operations when this workaround is employed. For 128KByte sequential IO on a 10G network, delayed ACKs reduced read throughput by 35%, from 910 MBytes/s (with delayed ACK disabled) to 588 MBytes/s (with delayed ACK enabled).

The packet traces used in our analysis were captured on the network between the ESXi host and the NFS server using an inline network tap and a hardware packet capture system. Packet traces collected in this way do not have capture drops and have accurate packet timestamps. These high-quality packet traces simplify the subsequent analysis.

To illustrate the TCP interaction between slow start and delayed ACK in ESXi NFS traffic, we examined a packet trace captured from an environment in which we observed the interaction. We present the annotated packet trace, followed by the Wireshark analysis steps we used to positively identify the presence of the TCP interaction in the packet trace.

A key lesson of this paper is that seemingly minor packet loss rates could have an outsized impact on overall performance for ESXi networked storage (NFS, iSCSI, vSAN). We recommend customers who are using ESXi networked storage and have a highly performance sensitive application to consider taking steps to identify and mitigate these undesirable interactions if appropriate.

> **See "Appendix: Relevant TCP Concepts" for more information about TCP concepts used in this paper, including duplicate ACK, fast retransmit, slow start, and delayed ACK.**

Notation for TCP Traffic

We use the following notation to describe TCP traffic between the two network endpoints X and Y of a TCP connection:

- X->Y
  - X and Y are TCP endpoints.
  - X sends a TCP segment to Y.
- X->Y, Seq= S
  - X sends a TCP segment with sequence number S to Y.
  - The TCP segment has a payload size of S.len bytes.
- Y->X, Ack= A
  - Y sends a TCP acknowledgement with acknowledgement number A to X.

Undesirable TCP Interaction: Slow Start and Delayed Acknowledgement

This section describes the TCP interaction that leads to poor ESXi NFS read performance.

Overview

In some packet loss scenarios, fast retransmit on the sender together with selective acknowledgement (SACK) on the receiver will quickly retransmit a missing TCP segment. For some sender TCP/IP stack implementations, however, the sender will enter the slow start state for subsequent TCP segments. If delayed ACK is enabled on the receiver, the ACK for the first slow start segment can be delayed by the full delayed ACK timer, which could be on the order of 100ms. The net effect is that a high bandwidth flow can be stalled for the delayed ACK timer interval (of 100ms) after each instance of packet loss, resulting in a significant drop in throughput.

TCP Slow Start and Delayed Acknowledgement

In this section, we describe how very minor packet loss on the network triggers an undesirable TCP interaction between slow start on the sender and delayed ACK on the receiver, resulting in poor data transfer throughput.

Consider the following data transfer scenario with sender X and receiver Y:

- X->Y: Data transfer, with minor packet loss
- X->Y: Fast retransmit
- X->Y: Slow start
- Y->X: Delayed ACK (T millisecond timer)

When there is packet loss on the network, each delayed ACK on receiver Y leads to a T millisecond (ms) pause in data transfer from X to Y. As T is typically multiple milliseconds (for ESXi's NFS client, T is 100ms) a very minor packet loss can result in very poor transfer throughput.

Both of the following preconditions are necessary for this undesirable TCP interaction to occur:

- Sender X's TCP/IP stack enters slow start state after fast retransmit
- Receiver Y's TCP/IP stack has enabled delayed ACK (of Tms)

If either of these preconditions is not satisfied, the undesirable interaction does not occur. This suggests that we can implement a workaround to suppress this interaction on either the sender or receiver side.

Detailed Scenario

In this section, we present a detailed scenario in which very minor packet loss leads to very poor data transfer throughput.

Consider the following data transfer scenario with sender X and receiver Y:

- Sender X sends 5 TCP segments (A,B,C,D,E) to receiver Y.
- The second segment (B) of these 5 TCP segments is lost.

Sender X sends 5 TCP segments (A,B,C,D,E) to receiver Y [1,2,3,4,5].

```
[1]  X->Y: Seq=A;  Y->X: Ack=B       [B==A+A.Len]
```

The second segment (B) was lost [2].

```
[2]  X->Y: Seq=B                     [Packet loss: B dropped]
```

The 3 trailing segments (C,D,E) after the lost segment (B) resulted in 3 duplicate acknowledgements (DUP ACKs) from Y [3,4,5].

```
[3]  X->Y: Seq=C;  Y->X: Ack=B       [Dup Ack=B #1]
[4]  X->Y: Seq=D;  Y->X: Ack=B       [Dup Ack=B #2]
[5]  X->Y: Seq=E;  Y->X: Ack=B       [Dup Ack=B #3]
```

X fast retransmits the missing segment B [6].

```
[6]  X->Y: Seq=B;  Y->X: Ack=E+E.Len [Fast retransmit of B: caught up.  Assume SACK]
```

X enters slow start state: it sends 1 segment (F) and waits for an ACK from Y [7]. Y waits for the Delayed ACK timer expiration (Tms) before sending the ACK [7].

```
[7]  X->Y: Seq=F                     [Slow start: Send 1 TCP segment]
     Y->X: Ack=F+F.Len               [Delayed ACK: after T ms]
```

X sends 2 segments (G, H) and waits for an ACK from Y [8,9].

```
[8]  X->Y: Seq=G
     X->Y: Seq=H                     [Slow start: Send 2 TCP segments]
```

Y immediately sends an ACK [9].

```
[9] Y->X: Ack=H+H.Len
```

X sends 4 segments (I, J, K, L) and waits for an ACK from Y [10, 11, 12, 13]. Y immediately sends an ACK [13].

```
[10] X->Y: Seq=I
[11] X->Y: Seq=J
[12] X->Y: Seq=K
[13] X->Y: Seq=L                    [Slow start: Send 4 TCP segments]
     Y->X: Ack=L+L.Len
```

In this scenario, each instance of packet loss in the X->Y direction (step [2]) triggers the slow start/delayed ACK interaction (step [7]). The Tms delayed ACK results in a Tms pause in data transfer. If T is large (say 100ms), even an extremely low packet loss rate could lead to very poor transfer throughput.

ESXi and NFS Server

In this section, we show how the slow start/delayed ACK interaction, described in the previous section, occurs in the context of ESXi NFS read operations from an NFS server.

For ESXi read operations from an NFS server:

- The sender (X) is the NFS server.
- The receiver (Y) is ESXi.
- ESXi NFS client with delayed ACK timer T=100ms.

In our scenario:

- MTU=9000: jumbo frames are enabled.
- TCP selective acknowledgement (SACK) is enabled.

NFS data transfer highlights:

- ESXi reads a large block (64KBytes+) from the NFS server.
- The NFS server sends 5 Maximum Segment Size (MSS-sized) TCP segments (A,B,C,D,E) to ESXi.
- One TCP segment (B) from the NFS server is lost.
- The NFS server fast retransmits the lost segment.
- The NFS server enters slow start state and sends one segment to ESXi.
- ESXi ACKs this single segment after a delay of 100ms. There is no data transfer during this 100ms interval.

NFS Data Transfer Details

> Note: Not all frames are shown.

```
Esx->NFSServer: NFS Read(64KBytes+) request
NFSServer->Esx: NFS Read reply, A
NFSServer->Esx: B [Minor packet loss: segment B lost]
NFSServer->Esx: C [Trailing segment(1)]
NFSServer->Esx: D [Trailing segment(2)]
NFSServer->Esx: E [Trailing segment(3)]
Esx->NFSServer: Triple Duplicate ACK [Triggered by Trailing segments(1,2,3)]
NFSServer->Esx: B [Fast retransmit of lost segment]
NFSServer->Esx: F [Slow start: Send 1 TCP segment]
Esx->NFSServer: Delayed ACK [100ms: long pause without data transfer]
NFSServer->Esx: G [Slow start: Send 2 TCP segments]
NFSServer->Esx: H
Esx->NFSServer: ACK [Immediate ACK]
NFSServer->Esx: I [Slow start: Send 4 TCP segments]
NFSServer->Esx: J
NFSServer->Esx: K
NFSServer->Esx: L
Esx->NFSServer: ACK [Immediate ACK]
```

- After segment B was lost, the 3 trailing segments from the NFS server (C,D,E) would each trigger one DUP ACK from ESXi. These 3 DUP ACKs together lead to a fast retransmission of B from the NFS server.
- After the NFS server fast retransmits the lost segment (B), it enters the slow start state. In this state, it sends one segment (F) and waits for an ACK from ESXi. (NFS server's slow start behavior depends on the implementation of its TCP/IP stack.) After ESXi receives the single segment F, it sends an ACK when its Delayed ACK timer expires (in 100ms). The slow start on NFS server and Delayed ACK on ESXi form a deadlock that is eventually broken by ESXi's Delayed ACK (100ms) timer. During any such 100ms interval, no data is being transferred from the NFS server to ESXi. These 100ms pauses in data transfer lead to poor NFS read throughput, even for a very low packet loss rate (of say 0.02%).

Signatures

The slow start/delayed ACK interaction for ESXi NFS traffic can be identified by 4 key signatures, in order:

1. Signature(1): Esx->NFSServer: Triple DUP ACK after packet loss
2. Signature(2): NFSServer->Esx: Fast retransmit of loss segment
3. Signature(3): NFSServer->Esx: Slow start, with one MSS-sized TCP segment in flight
4. Signature(4): Esx->NFSServer: Delayed ACK, after 100ms

In particular, the 100ms delay between Signature(3) and Signature(4) is the distinguishing feature of the undesirable TCP interaction.

Wireshark Analysis

In the following two sections, we demonstrate two alternative approaches of using Wireshark to identify these signatures: heuristic and comprehensive.

Heuristic Identification

In the heuristic approach, we use a Wireshark graph to identify the key signature of the undesirable TCP interaction—100ms pauses in data transfer. This approach is relatively easy to follow, even for users not experienced with packet analysis using Wireshark.

We analyze Wireshark's tcptrace graph, which shows the TCP sequence number vs time progression in a data transfer, in the NFS Server to ESXi direction.

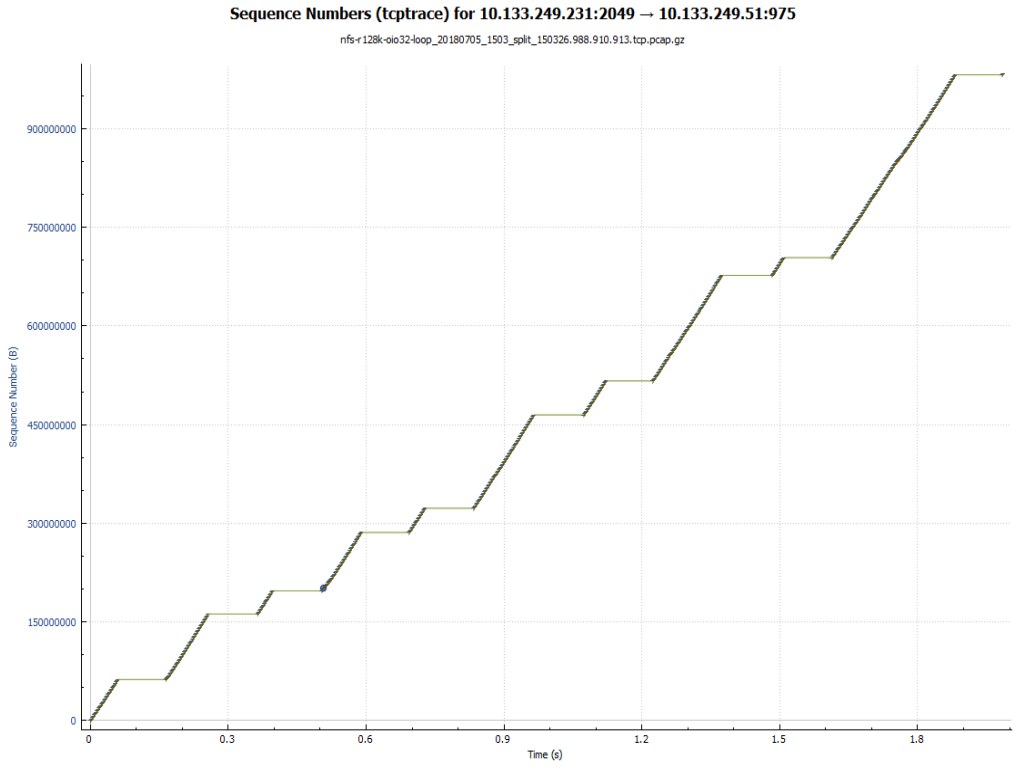

Figure 1. Example of tcptrace graph in Wireshark showing evidence of slow start/delayed ACK TCP interaction

Each horizontal line segment in Figure 1 corresponds to a 100ms pause (caused by ESXi delayed ACK) with no data transfer.

If there are 100ms data transfer pauses in the tcptrace graph, we can reasonably infer that the slow start/delayed ACK TCP interaction is present in the packet trace.

Comprehensive Identification

In the comprehensive approach we explicitly identify all 4 signatures of the TCP interaction described in the previous section, "Signatures." This approach is more suitable for users who are experienced with packet analysis using Wireshark.

Details of the sample packet trace referenced here can be found in "Packet Trace File Stats."

Using Wireshark, we identify the 4 signatures of the slow start/delayed ACK interaction in the following order:

1. Identify Signature(4) Esx->NFSServer, Delayed ACKs with 100+ms delta time relative to the previous frame.
   - Wireshark Display Filter:

```
(tcp.dstport==2049) && (tcp.flags.ack==1) && (frame.time_delta >= 0.1)
```

   - 0.1s = 100ms
   - For sample packet trace: Frame= 614 388

2. Identify Signature(3) NFSServer->Esx, Slow Start with only 1 segment in flight
   - The frame for Signature(3) should immediately precede the frame for Signature(4)
   - Wireshark Display Filter:

```
(tcp.srcport==2049) && (tcp.analysis.bytes_in_flight==8948)
```

   - With jumbo frames, an MSS-sized TCP segment has 8948 bytes of payload.
   - For sample packet trace: Frame= 614 387

3. Identify Signature(2) NFSServer->Esx, Fast Retransmit
   - Wireshark Display Filter:

```
(tcp.srcport==2049) && (tcp.analysis.fast_retransmission)
```

   - Sequence number of TCP segment: Sx [For next step: Signature (1)]
   - For sample packet trace: Frame= 614 377 with Sequence number= 685 736 354

4. Identify Signature(1) Esx->NFSServer, 3rd DUP ACK
   - Wireshark Display Filter:

```
(tcp.ack==Sx) && (tcp.analysis.duplicate_ack_num==3)
```

   - For sample packet trace: Frame= 614 313 with ACK number= 685 736 354, which matches the sequence number (Sx) from Signature(2)

If all 4 signatures are identified, we have positively confirmed the presence of the slow start/delayed ACK interaction in the packet trace.

Workaround: Disable ESXi NFS Client Delayed ACK

If the slow start/delayed ACK interaction is identified, you should disable ESXi's delayed ACK for NFS to suppress the interaction. This could lead to significantly improved performance. This workaround is available in vSphere 7.0.

To disable the ESXi NFS client's TCP delayed ACK (100ms):

1. Enter the following command from the ESXi console:

```
esxcli system settings advanced set -o "/SunRPC/SetNoDelayedAck" -i 1
```

2. Verify SetNoDelayedAck was successfully changed:

```
esxcli system settings advanced list | grep -A10 /SunRPC/SetNoDelayedAck
```

See KB 59548 for details [2].

Example: ESXi NFS Experiment

To illustrate the slow start–delayed ACK TCP interaction in ESXi NFS traffic, we describe an experiment to reproduce the undesirable TCP interaction between an ESXi host and an NFS server. We captured network packet traces in an environment where we observed the interaction for subsequent Wireshark analysis.

Experiment Setup

We connected an ESXi host to an NFS datastore over a 10G network. We then configured a VM on the ESXi host with a vdisk backed by a vmdk file on the NFS datastore.

We ran Iometer in the VM to generate sequential read IOs of 128 kilobytes (128KBytes) with 32 outstanding IOs (OIO=32) to the vdisk.

Key Components

We used the following components for the experiment setup.

Component	Type or Value
vSphere/ESXi	7.0 internal build
NFS server	Commercial product
Network	10G, MTU=9000
Protocol	NFSv3
I/O	Iometer, seq read (128KB), OIO=32, Worker=1
ESXi NFS client delayed ACK	Enabled (default); Disabled (workaround)

Table 1. Key components of the experiment setup

ESXi NFS Client: Delayed ACK	Undesirable Interaction	Seq Read (128KBytes) OIO=32 Throughput (MBytes/second)
Enabled (default)	Yes	588
Disabled (workaround)	No	910

Table 2. Read throughput: Enabled/Disabled delayed ACK

Packet Capture

The network traffic between the ESXi host and the NFS server was captured using a NetOptics inline optical network tap and an Fmad Engineering Fmadio20 hardware packet capture system. The hardware packet capture system ensured there were no capture drops, and that packets were timestamped accurately to nanosecond resolution.

- Physical network
  - 10G, optical
  - LAN, not WAN, latencies
- Physical network topology:

```
ESXi --- [inline network TAP] --- 10G switch --- NFS server
```

- Full packets—including TCP payload—are captured.

> **On 10G and faster networks, a packet trace captured on a network endpoint—say on ESXi using pktcap-uw—will most likely have a significant number of capture drops. These capture drops will make the analysis of a packet loss–related problem extremely challenging.**

For more information on packet capture, see ESX IP Storage Troubleshooting Best Practice: Packet Capture and Analysis at 10G [3].

Packet Trace File Statistics

Statistics for the packet trace file are as follows:

Parameter	Value	Info
Frames	796380	
Time Duration	66 sec	
File Size	109MB	full packets (including payload) in .pcap.gz format

Table 3. Statistics for packet trace file

Wireshark TCP analysis statistics (using tcp.analysis.* display filters):

Wireshark Display Filter	Value
tcp.analysis.flags	2649 (0.33%)
tcp.analysis.duplicate_ack	2470 (0.31%)
tcp.analysis.fast_retransmission	165 (0.02%)
tcp.analysis.retransmission	166 (0.02%)

Table 4. Wireshark TCP analysis statistics

Wireshark TCP analysis statistics for this packet trace file suggests that the packet trace is relatively clean overall and would not generally deserve a detailed (frame-by-frame) analysis. We could have easily reached the incorrect conclusion that packet loss is not a big issue. Without a detailed Wireshark analysis, the undesirable slow start/delayed ACK interaction would have remained undetected.

Annotated Packet Trace

In this section, we present an annotated version of the packet trace file that was captured during the experiment. Key signatures of the undesirable interaction—packet loss, triple DUP ACK, fast retransmit, slow start, and delayed ACK—are identified (by "*").

Highlights

> **Note: Not all frames in packet trace are shown. Storage is NFS Server.**

```
Frame    Src/Dst              Details                        Comments
614 291  Storage->Esx         Seq=685 718 458, L=8948
614 296  Storage->Esx         Seq=685 736 354, L=8948        [Packet loss] *
614 297  Storage->Esx         Seq=685 745 302, L=8948        [Trailing TCP segment: 1]

614 301  Esx->Storage         Ack=685 718 458
614 303  Esx->Storage         Ack=685 736 354                [Frame 614296 dropped]

614 304  Storage->Esx         Seq=685 790 042, L=8948        [Trailing TCP segment: 2]

614 305  Esx->Storage         Ack=685 736 354                [DUP ACK 614303#1] *

614 306  Storage->Esx         Seq=685 798 990, L=8948        [Trailing TCP segment: 3]

614 309  Esx->Storage         Ack=685 736 354                [DUP ACK 614303#2] *
614 313  Esx->Storage         Ack=685 736 354                [DUP ACK 614303#3: Triple DUP ACK] *

614 375  Esx->Storage         Ack=685 736 354                [DUP ACK 614303#21]

614 377  Storage->Esx         Seq=685 736 354, L=8948        [Fast Retransmit of 614296] *
614 380  Esx->Storage         Ack=686 255 338                [Caught up]

614 387  Storage->Esx                                        [Slow start: Send 1 segment] *
614 388  Esx->Storage         DeltaTime=102ms                [Delayed ACK (100ms)] *

614 389  Storage->Esx                                        [Send 2 segments]
614 392  Storage->Esx                                        [Send 4 segments]
614 397  Storage->Esx                                        [Send 5 segments]
614 403  Storage->Esx                                        [Send 6 segments]
614 412  Storage->Esx                                        [Send 8 segments]
```

More Details

Note: Not all frames in packet trace are shown.

```
[Fast Retransmit]
Frame    Src/Dst              Details                        Comments
614 291  Storage->Esx         Seq=685 718 458, L=8948
614 296  Storage->Esx         Seq=685 736 354, L=8948        [Packet loss] *
614 297  Storage->Esx         Seq=685 745 302, L=8948        [Trailing TCP segment: 1]
614 301  Esx->Storage         Ack=685 718 458
614 303  Esx->Storage         Ack=685 736 354                [Frame 614296 dropped]
614 304  Storage->Esx         Seq=685 790 042, L=8948        [Trailing TCP segment: 2]
614 305  Esx->Storage         Ack=685 736 354                [DupACK 614303#1] *
614 306  Storage->Esx         Seq=685 798 990, L=8948        [Trailing TCP segment: 3]
614 309  Esx->Storage         Ack=685 736 354                [DupACK 614303#2] *
614 313  Esx->Storage         Ack=685 736 354                [DupACK 614303#3: Triple DUP ACK] *
614 375  Esx->Storage         Ack=685 736 354                [DupACK 614303#21]
614 377  Storage->Esx         Seq=685 736 354, L=8948        [Fast Retransmit of 614296] *
614 378  Esx->Storage         Ack=685 736 354                [DupACK 614303#22]
614 379  Esx->Storage         Ack=685 736 354                [DupACK 614303#23]
614 380  Esx->Storage         Ack=686 255 338                [Caught up]

[Slow Start and Delayed ACK]
Frame    Src/Dst              Details                        Comments
614 387  Storage->Esx         Seq=686 255 338, L=8948        [Slow start: Send 1 segment] *
614 388  Esx->Storage         Ack=686 264 286                [DelayedACK: DeltaTime=102ms] *
614 389  Storage->Esx         Seq=686 264 286, L=8948        [Send 2 segments]
614 390  Storage->Esx         Seq=686 273 234, L=8948
614 391  Esx->Storage         Ack=686 282 182
614 392  Storage->Esx         Seq=686 282 182, L=8948        [Send 4 segments]
614 393  Storage->Esx         Seq=686 291 130, L=8948
614 394  Storage->Esx         Seq=686 300 078, L=8948
614 395  Storage->Esx         Seq=686 309 026, L=8948
614 396  Esx->Storage         Ack=686 309 026
614 397  Storage->Esx         Seq=686 317 974, L=8949        [Send 5 segments]
614 398  Storage->Esx         Seq=686 326 922, L=8949
614 399  Storage->Esx         Seq=686 335 870, L=8948
614 400  Storage->Esx         Seq=686 344 818, L=8948
614 401  Storage->Esx         Seq=686 353 766, L=8948
614 402  Esx->Storage         Ack=686 344 818
614 403  Storage->Esx         Seq=686 362 714, L=8948        [Send 6 segments]
614 404  Storage->Esx         Seq=686 371 662, L=8948
614 405  Storage->Esx         Seq=686 380 610, L=8948
614 406  Storage->Esx         Seq=686 389 558, L=8948
614 407  Storage->Esx         Seq=686 398 506, L=8948
614 408  Storage->Esx         Seq=686 407 454, L=8948
614 409  Esx->Storage         Ack=686 371 662
614 410  Esx->Storage         Ack=686 398 506
614 411  Esx->Storage         Ack=686 416 402
614 412  Storage->Esx         Seq=686 416 402, L=8948        [Send 8 segments]
614 413  Storage->Esx         Seq=686 425 350, L=8948
614 414  Storage->Esx         Seq=686 434 298, L=8948
614 415  Storage->Esx         Seq=686 443 246, L=8948
614 416  Storage->Esx         Seq=686 452 194, L=8948
614 417  Storage->Esx         Seq=686 461 142, L=8948
614 418  Storage->Esx         Seq=686 470 090, L=8948
614 419  Storage->Esx         Seq=686 479 038, L=8948
614 420  Esx->Storage         Ack=686 443 246
```

Wireshark Analysis

Heuristic Identification

In the heuristic approach, we examine Wireshark's tcptrace graph (in the NFS Server to ESXi direction).

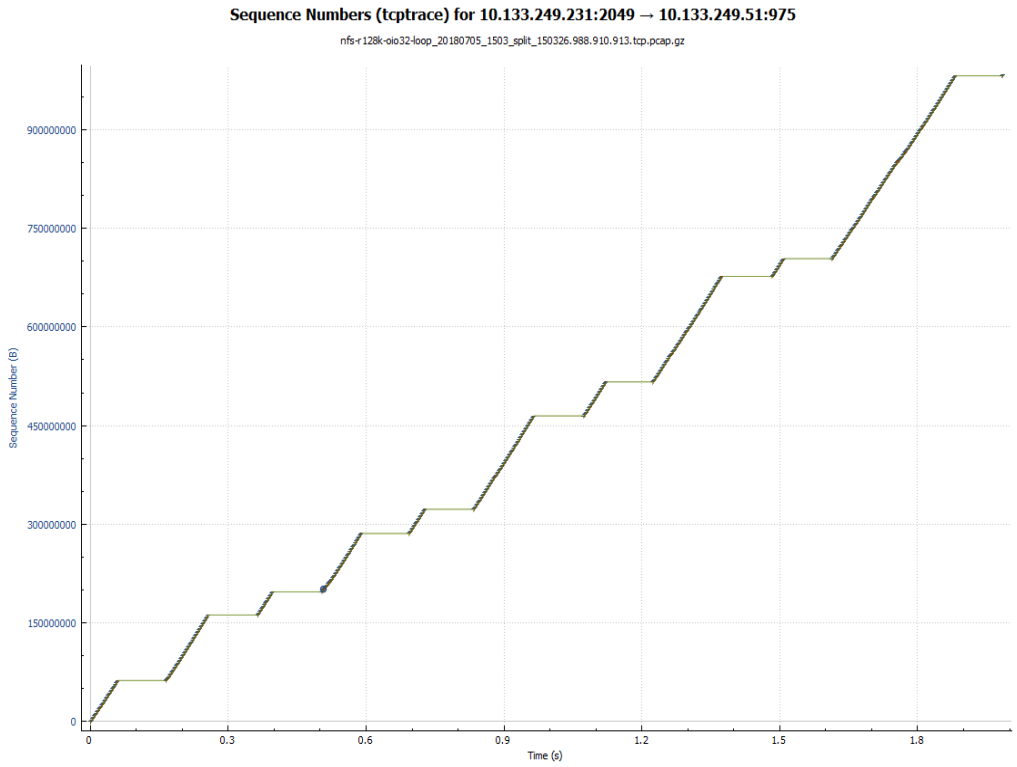

Figure 2. Through Wireshark graph analysis, we observe 100ms pauses, from which we can infer a slow start/delayed ACK

Each horizontal line segment in Figure 2 corresponds to a 100ms pause (caused by ESXi delayed ACK) with no data transfer. Based on these 100ms pauses, we can infer the presence of slow start/delayed ACK interactions in the packet trace.

Comprehensive Identification

In the comprehensive approach, we identify the 4 signatures of the slow start/delayed ACK interaction using Wireshark. (Also see "Signatures.")

These signatures are identified in the following order:

1. Identify Signature(4) Esx->NFSServer, Delayed ACKs with 100+ms delta time relative to the previous frame
   - Wireshark display filter:

```
(tcp.dstport==2049) && (tcp.flags.ack==1) && (frame.time_delta >= 0.1)
```

   - 0.1s = 100ms
   - Frame= 614 388

2. Identify Signature(3) NFSServer->ESXi, slow start with only 1 segment in flight:
   - The frame for Signature(3) should immediately precede the frame for Signature(4)
   - Wireshark Display Filter:

```
(tcp.srcport==2049) && (tcp.analysis.bytes_in_flight==8948)
```

   - With jumbo frames, an MSS-sized TCP segment has 8948 bytes of payload.
   - Frame= 614 387

3. Identify Signature(2) NFSServer->Esx, Fast Retransmit
   - Wireshark display filter:

```
(tcp.srcport==2049) && (tcp.analysis.fast_retransmission)
```

   - Sequence number of TCP segment: Sx [For next step: Signature(1)]
   - Frame= 614 377 with sequence number= 685 736 354

4. Identify Signature(1) Esx->NFSServer, 3rd DUP ACK
   - Wireshark display filter:

```
(tcp.ack==Sx) && (tcp.analysis.duplicate_ack_num==3)
```

   - Frame= 614 313 with ACK number= 685 736 354, which matches sequence number from Signature(2)

When all 4 signatures of the slow start/delayed ACK interaction have been identified, we can confirm the presence of the undesirable TCP interaction in the packet trace.

Summary

In this example, we analyzed ESXi NFS read traffic from an NFS datastore, with very minor packet loss (of 0.02%) of TCP segments from NFS datastore to ESXi.

When such packet loss occurs, fast retransmit and selective acknowledgement (SACK) will quickly retransmit the missing TCP segment. However, for some sender TCP/IP stack implementations–including the TCP/IP stack on this particular NFS datastore–the sender will enter the slow start state. As Delayed ACK is enabled (by default) on the receiver ESXi, the ACK for the first slow start segment is delayed for the delayed ACK interval of 100ms. The net effect is that the high bandwidth TCP flow of NFS read traffic is stalled for 100ms after each instance of packet loss, resulting in a significant 35% drop in NFS read throughput.

Conclusion

In this paper, we examined how very minor packet loss (of as low as 0.02%) on the network could trigger an undesirable TCP interaction between slow start on a sender and delayed acknowledgement on the receiver, leading to unexpectedly poor data transfer throughput. We analyzed a scenario in which this interaction manifests itself in the context of ESXi read operations from an NFS server. The 100ms delayed acknowledgements on ESXi lead to 100ms pauses in data transfer, resulting in poor NFS read throughput.

To determine if this undesirable TCP interaction is present in a given environment, we captured and analyzed the network traffic between ESXi and the NFS server. We demonstrated how key signatures of this interaction can be identified using Wireshark analysis. If the undesirable interaction is found to be present, it can be suppressed by disabling TCP delayed ACKs on the ESXi NFS client. In our experiments, we observed significant throughput improvements for NFS large IO-sized read operations when this workaround is employed.

The packet traces used in our analysis were captured on the physical network between ESXi and the NFS server using an inline network tap and a hardware packet capture system. Packet traces thus collected do not have capture drops and have accurate packet timestamps. These high-quality packet traces greatly simplify the subsequent analysis.

To illustrate the TCP interaction between slow start and delayed ACK in ESXi NFS traffic, we examined a packet trace captured from an environment in which the interaction was observed. We presented the annotated packet trace, followed by the Wireshark analysis steps used to positively identify the presence of the TCP interaction in the packet trace.

A key lesson of this paper is that seemingly minor packet loss rates could have an outsized impact on overall performance for ESXi networked storage (NFS, iSCSI, vSAN). We recommend customers who are using ESXi networked storage and have a highly performance-sensitive workload to consider taking steps to identify and mitigate these undesirable interactions if appropriate.

Appendix: Relevant TCP Concepts

This section contains a very brief review of the relevant TCP concepts that are key to understanding the undesirable TCP interactions between ESXi and some NFS servers:

- Sequence number, acknowledgement number, and ACK rules
- Packet loss: Duplicate acknowledgements and fast retransmit
- Congestion control: Slow start

For more information on these topics, see the presentations from recent SharkFest Conferences [4].

Sequence Number, Acknowledgement Number, and ACK Rules

- **Sequence number:** TCP assigns a sequence number to each data byte being sent. The sequence number of a TCP segment is the sequence number of the first data byte in the segment.
- **Acknowledgement (ACK) number:** TCP uses an ACK number to track the bytes that have been received. A receiver sends an acknowledgement with ACK number A to inform the sender that all bytes with sequence number 0..A-1 have been received, and that the next expected sequence number from the sender is A.
- **TCP ACK rules:**
  - If a receiver receives 1 TCP segment from the sender, it may wait up to delayed ACK timer expiration (ESXi: 100ms) before sending an ACK.
  - If a receiver receives 2 TCP segments from the sender, it must send an ACK immediately.

Packet Loss: Duplicate Acknowledgements (DUP ACKs) and Fast Retransmit

TCP uses two mechanisms to ensure retransmission if data is lost.

- Retransmission timeout (RTO): slow mechanism
- Triple DUP ACKs and fast retransmit: fast mechanism

Retransmission Timeout

Sender X sends a TCP segment to receiver Y and sets an RTO timer. If X has not received an ACK for this segment from Y upon RTO timer expiration, X retransmits the unacknowledged (and presumably lost) segment to Y.

RTO does not play any part in the TCP interactions of interest in this paper.

Triple DUP ACKs and Fast Retransmit

- DUP ACK:
  - TCP acknowledges incoming segments in order.
  - If a receiver sends an ACK with ACK number A and then receives a segment with sequence number A+Z from the sender, it will generate a duplicate ACK with ACK number A to inform the sender that the segment with sequence number A has not been received. We will not cover selective acknowledgement (SACK) here.

- Fast retransmit rule:
  - If a sender receives 3 duplicate acknowledgements (DUP ACKs) for a missing segment, it retransmits the missing segment immediately.
  - Fast retransmit only happens if the sender is sending 3+ trailing segments after the missing segment. If there are fewer than 3 trailing segments, the missing segment will be retransmitted when the RTO timer expires.
  - Triple DUP ACKs and fast retransmit play a role in the TCP interactions of interest in this paper.

Congestion Control: Slow Start

- TCP assumes that packet loss is caused by network congestion. When packet loss is detected at the sender (due to triple DUP ACKs from the receiver), the sender may reduce the sending rate by entering the slow start state. This behavior depends on the sender's TCP/IP stack implementation.
- If a sender receives 3+ DUP ACKs from the receiver:
  - Sender retransmits the missing segment immediately.
  - Sender **may** enter slow start, depending on TCP/IP stack implementation.
- If a sender enters the slow start state, the following interaction between sender and receiver will be observed (Sender: TCP Tahoe implementation):
  1. Sender sends one segment and waits for an ACK.
  2. Receiver waits for delayed ACK timer expiration before sending ACK.
  3. When sender receives the ACK, it sends 2 segments and waits for an ACK.
  4. Receiver sends ACK immediately.
  5. When sender receives the ACK, it sends 4 segments and waits for an ACK.
  6. Receiver sends ACK immediately.

The data transfer rate from the sender increases exponentially during slow start until some limit is reached. This limit is not relevant to the discussion in this paper.

References

[1] VMware. (2018, October) ESX/ESXi hosts might experience read or write performance issues with certain storage arrays (1002598). [Online]. https://kb.vmware.com/s/article/1002598

[2] VMware. (2019, November) NFS poor read throughput when there is ESX packet loss: NFS server slow start interacts with ESX delayed ACK (59548). [Online]. https://kb.vmware.com/s/article/59548

[3] Kinson Ho. (2017, December) ESX IP Storage Troubleshooting Best Practice: Packet Capture and Analysis at 10G. [Online]. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/performance/ESX-IP-storage-troubleshooting.pdf

[4] Wireshark.org. (2019) SharkFest. [Online]. https://sharkfestus.wireshark.org/sf19

Author Info

Kinson Ho is a staff engineer in Performance Engineering at VMware with a focus on vSphere networked storage performance, including NFS, iSCSI, and vSAN File Services performance. His interests include the use of packet capture and analysis for performance troubleshooting and optimization on high speed networks.

Acknowledgements

The author thanks Steve Faulkner, Chien-Chia Chen, and Shahnawaz Shahpahelwan for their highly insightful comments about the paper, and Julie Brodeur for superb technical editing that greatly improved the organization and readability of the paper.

Special thanks are due to Hansang Bae, Jasper Bongertz, and Christian Landstrom for introducing me to the art of packet trace analysis in their excellent SharkFest presentations, and to Aaron Foo for expeditious responses to my numerous questions regarding the Fmad packet capture system.