MITRE

Improving TCP Throughput for AIX Hosts Communicating Over High Latency Network Connections

McLean, VA

Author(s):
Neil A. Kirr
Paul E. Nguyen

February 2021

Abstract

This document describes a method for improving the Transmission Control Protocol (TCP) throughput between AIX hosts communicating over a high latency network connection (e.g., a wide area network (WAN)). MITRE observed that the TCP throughput between AIX hosts communicating across a high bandwidth WAN was low while that of Linux hosts was significantly higher. While researching this issue, MITRE learned that AIX uses the NewReno TCP congestion algorithm, and that Red Hat Enterprise Linux (RHEL) uses the Cubic TCP congestion algorithm. These algorithms are responsible for the observed difference in TCP throughput across the WAN. To improve the throughput for AIX hosts, a method was developed using RHEL hosts and a utility called “socat” that allows the Linux hosts to act as TCP proxies for the AIX hosts’ WAN communications. Since Linux uses the Cubic algorithm for TCP congestion control, the overall TCP throughput is improved over NewReno. MITRE observed between 7 and 10 times more throughput between AIX hosts using this method.

Executive Summary

Different operating systems utilize different TCP congestion control algorithms. For AIX, the algorithm is “NewReno” and for Linux, the algorithm is “Cubic”. MITRE observed that the TCP throughput of the NewReno algorithm was much lower than expected between two 10 Gbps ethernet connected hosts communicating across a 100 Gbps WAN with 63 milliseconds (ms) of network latency. The average observed throughput was only 3.6 MBps (28.8 Mbps).

This document provides a solution that greatly improves the TCP throughput between AIX hosts communicating over a high latency network connection. The solution requires the use of Red Hat Enterprise Linux (RHEL) hosts and a utility called *socat* to send data across the WAN on behalf of the AIX hosts. The test environment used the IBM PowerPC 64-bit Little Endian (PPC64LE) version of RHEL 7.6 running on IBM POWER9 servers. While other hosting platforms were not tested, this solution does not preclude the use of alternative platforms to run the RHEL hosts (e.g., RHEL x64).

Using the configuration as outlined in this document, data throughput over a single TCP socket between AIX hosts was <u>increased approximately 7 to 10 times</u>. In addition, a major advantage of this solution is that most environments should be able to implement it with little to no additional cost. The MITRE team implemented this solution using existing hardware and software resulting in no additional cost.

Table of Contents

List of Figures

List of Tables

1 Low TCP Throughput

This section provides an overview of the low Transmission Control Protocol (TCP) throughput between AIX hosts as observed by the MITRE team. The issue was first seen by the operations team while transferring data between sites connected by a wide area network (WAN). All forms of communication were affected: file transfers regardless of protocol used, database replications, etc. Large data transfers of multiple terabytes (TB) between AIX hosts across the WAN were taking days, instead of hours, to complete. However, data transfers between AIX hosts on the Local Area Network (LAN) did not experience low TCP throughput, and TB of data could be transferred in minutes to hours.

The MITRE team collaborated with the operations team and IBM engineers to determine the cause of the low TCP throughput across the WAN.

1.1 Summary

Figure 1 summarizes the low TCP throughput as observed between AIX hosts over a high latency network connection:

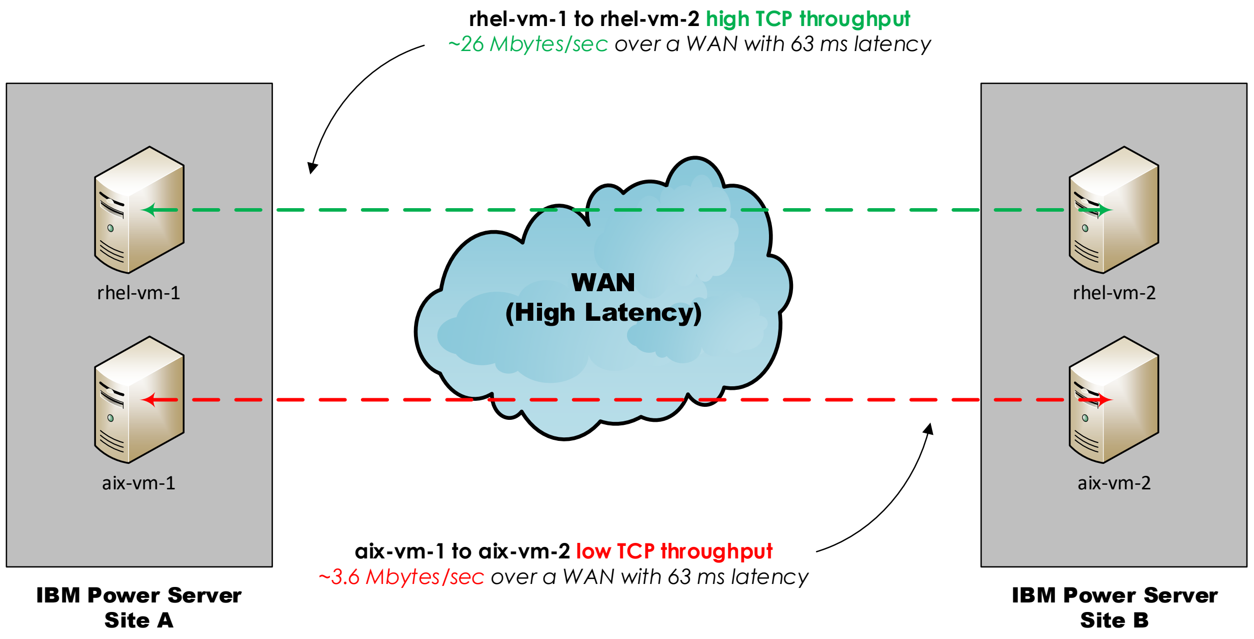
rhel-vm-1 to rhel-vm-2 high TCP throughput
~26 Mbytes/sec over a WAN with 63 ms latency
rhel-vm-1
aix-vm-1
WAN
(High Latency)
rhel-vm-2
aix-vm-2
aix-vm-1 to aix-vm-2 low TCP throughput
~3.6 Mbytes/sec over a WAN with 63 ms latency
IBM Power Server
Site A
IBM Power Server
Site B

Figure 1: Low TCP Throughput between AIX Hosts

As seen in **Figure 1**, the data transfers between AIX hosts were observed to be limited to approximately 3.6 MBps over a WAN connection with 63 ms of latency. At this rate, data transfers across the WAN are intolerably slow for production operations. For example, a transfer of 2 TB of data using a single TCP socket at this rate (assuming a constant rate) would take 7 days and 10 hours to complete. Replicating a large database across the WAN is an example of a process that would suffer greatly from this throughput rate.

An additional observation was that virtual machines (VMs) running the Red Hat Enterprise Linux (RHEL) OS on the IBM POWER9 exhibited acceptable TCP throughput for data transfers across the WAN. During testing, the transfer of a 1 gigabyte (GB) test file was approximately 7 to 10 times faster between the RHEL hosts as compared to AIX hosts on the same POWER9 servers. This was notable as the RHEL VMs were using the same servers, storage, and networking as the AIX VMs.

While collaborating with IBM on the TCP throughput issue, the MITRE team learned that the cause of the low throughput is the NewReno congestion algorithm that is used by the AIX TCP/IP stack. The network congestion algorithm determines the rate at which the host should transmit data to avoid overwhelming the network. In this case the observation is that the NewReno algorithm sends data at a lower rate than the Cubic algorithm on the same WAN. Note that as the algorithm itself is the cause of the low throughput, there is nothing to patch or upgrade on the AIX host.

Since the NewReno algorithm is unchangeable, the MITRE team evaluated other ways to increase the AIX data transfer rates across the WAN.

2 An Observation and an Idea

The most important data point available during the troubleshooting of this issue was the performance of the RHEL hosts. Data transfers between these hosts was approximately 7 to 10 times faster than the AIX hosts using the same infrastructure. This sparked an idea:

Could the higher TCP throughput of RHEL hosts be used to improve TCP throughput for AIX hosts communicating across the WAN?

2.1 Test Environment Overview

Initially, the idea was to create secure shell (SSH) tunnels to forward traffic between two data centers across the WAN – from *Site A* to *Site B*. While this concept worked and did improve TCP throughput by approximately 4 to 5 times in the test environment, it also added significant overhead to the connectivity in the form of encryption, SSH protocol performance limitations, and authentication/key management complexities.

Thus, the MITRE team searched for another method to create a "TCP port proxy" that did not have the added overhead of the SSH tunnels. The search resulted in the discovery of an open-source Linux utility known as "*socat"*. The *socat* utility is used to relay TCP network traffic between hosts and is readily available on RHEL and other Linux distributions, including RHEL for IBM Power systems (RHEL PPC64LE). The MITRE team used this distribution to test the concept.

With socat in hand, the MITRE team created a test environment as shown in **Figure 2**:

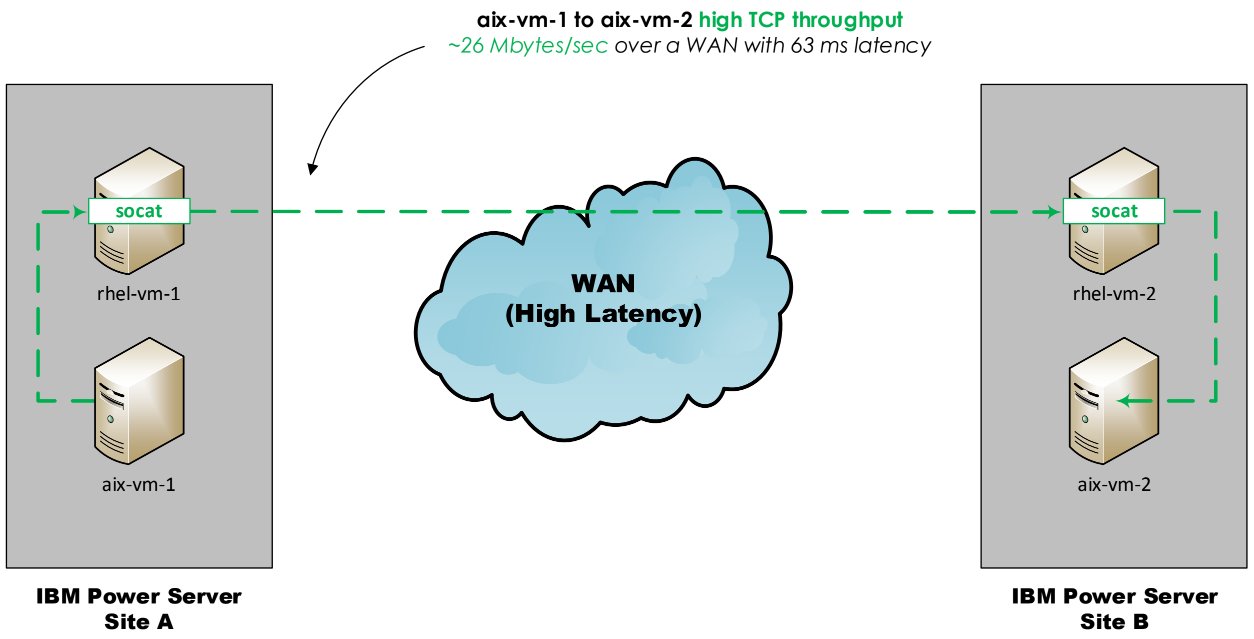

Figure 2: An Overview of the socat Test Environment

The goal was to have the RHEL hosts send data across the WAN on behalf of the AIX hosts. This seemed logical since the TCP throughput for WAN communications of the Cubic congestion algorithm was observed to be approximately 7 to 10 times faster than NewReno. The test configuration leveraged the hosts in Figure 2 in the following manner:

- ***aix-vm-1* sends data to *rhel-vm-1* via the LAN at Site A.**
  aix-vm-1 is the source host. Since the LAN has little to no latency, the NewReno algorithm sends data quickly, and the TCP throughput to *rhel-vm-1* is very high (the MITRE team observed a LAN throughput of approximately 1550 MBps in the test environment).

- ***rhel-vm-1* sends data to *rhel-vm-2* across the WAN**.
  This transfer takes advantage of the higher TCP throughput of the Cubic congestion algorithm.

- ***rhel-vm-2* sends data to *aix-vm-2* via the LAN at Site B**.
  aix-vm-2 is the destination host.

In this way, data is transferred from *aix-vm-1* to *aix-vm-2* with higher TCP throughput as compared to having the AIX hosts communicate directly. Thus, the key is to avoid the NewReno congestion algorithm for WAN communications.

The socat processes on *rhel-vm-1* and *rhel-vm-2* essentially create a “TCP port tunnel”. The proxied TCP connection takes advantage of the high TCP throughput of the Cubic algorithm for WAN communications, as well as the high throughput the NewReno algorithm for LAN communications. This configuration results in a TCP throughput that is significantly higher than a direct AIX-to-AIX connection. The MITRE team observed an increase in TCP socket throughput between AIX hosts across the WAN of approximately 7 to 10 times for secure file transfers using port tcp/22 and even higher rates for iPerf tests.

Using this method, the MITRE team observed the throughput increase from approximately 3.6 MBps to approximately 26 MBps for a single socket across the WAN. Note that the test environment used an encrypted WAN connection, so the observed throughput includes the overhead of data encryption and decryption at the network layer between the sites.

3 The Details

This section provides the technical details for the implementation of the socat test environment. This technique can be used to create a similar setup for any environment.

3.1 Pre-Requisites

The minimum pre-requisites are:

- One RHEL host running at "Site A"
- One RHEL host running at "Site B"
- socat utility installed on both RHEL hosts

Note that the RHEL hosts used in the test environment were PPC64LE VMs running on the IBM POWER9 servers, but this same technique could be done using RHEL hosts running on an Intel platform. It is also possible that other Linux distributions (e.g., SUSE, Ubuntu, etc.) could serve as TCP port proxies, however these configurations were not tested by the MITRE team and are thus not specifically recommended.

3.2 An Example Configuration

For this example, a tunnel was created for an SSH (SCP or SFTP) file transfer. The overall data flow is from *aix-vm-1* to *aix-vm-2*. If the data flow went from *aix-vm-2* to *aix-vm-1*, the tunnel would simply be built in reverse.

The specific port used for this example (6000) was selected arbitrarily. Any TCP port can be used. Keep in mind that each tunnel endpoint will require a unique port on the RHEL hosts.

For this example, the hosts have the following IP addresses:

Table 1: Hostnames and IP Addresses for Example Configuration

Hostname	IP Address
aix-vm-1	10.10.10.10
rhel-vm-1	10.10.10.20
aix-vm-2	10.20.20.10
rhel-vm-2	10.20.20.20

To create the tunnel, run the following commands:

3.2.1 On *rhel-vm-1* (10.10.10.20):

Perform the following steps on this host:

- Create the script "socat-tunnel.sh" from **Appendix A.1**
- Create a service named "socat-tunnel-6000-to-aix-vm-2-22.service" using the script and instructions in **Appendix A.2**. Substitute the values as shown below:
  - `X = 6000`
  - `B = aix-vm-2`
  - `Y = 22`
  - `b = 10.20.20.20`
  - `y = 6000`
- Get the status of the TCP port tunnel process:

  ```
  systemctl status socat-tunnel-6000-to-aix-vm-2-22
  ```

Try killing the socat process (`ps -ef | grep socat`, take note of the `PID`, then `kill -9 PID`) and ensure that it respawns automatically.

3.2.2 On *rhel-vm-2* (10.20.20.20):

Perform the following steps on this host:

- Create the script "socat-tunnel.sh" from **Appendix A.1**
- Create a service named "socat-tunnel-6000-to-aix-vm-2-22.service" using the script and instructions in **Appendix A.2**. Substitute the values as shown below:

  a. `X = 6000`

  b. `B = aix-vm-2`

  c. `Y = 22`

  d. `b = 10.20.20.10`

  e. `Y = 22`

- Get the status of the TCP port tunnel process:

  ```
  systemctl status socat-tunnel-6000-to-aix-vm-2-22
  ```

Try killing the socat process (`ps -ef | grep socat`, take note of the `PID`, then `kill -9 PID`) and ensure that it respawns automatically.

3.2.3 Test Environment Configuration Details

Figure 3 shows the details of the completed tunnel:

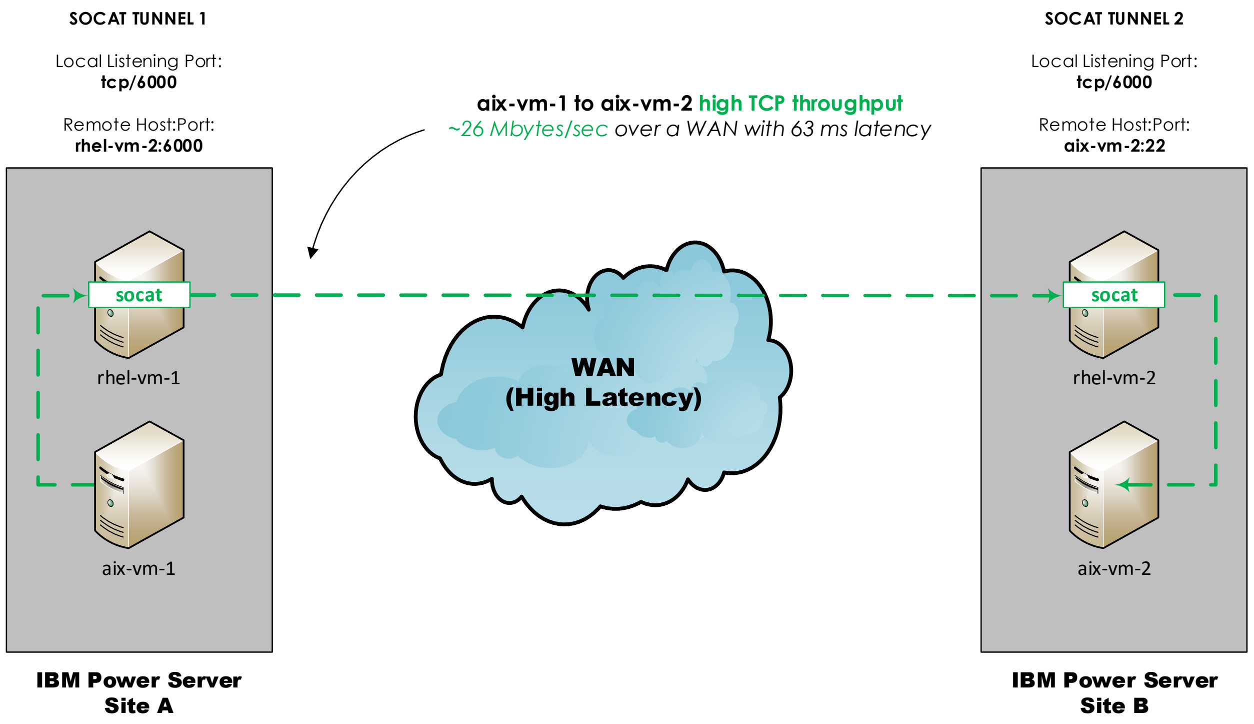

Figure 3: Test Environment Configuration Details

Note that as configured here, any host at "Site A" can use the tunnel since it is simply listening on *rhel-vm-1*:*6000* for incoming packets. See **Appendix A.3** for an alternative service version that limits the incoming connections to specific source IP addresses. Alternatively, the firewall service (`firewalld`) on *rhel-vm-1* can be configured to restrict access to that port to specific IP addresses, but that is beyond the scope of this document.

To use the tunnel, run any command like the following on host *aix-vm-1*:

```
scp -P 6000 my.data rhel-vm-1:/tmp/
```

Host *aix-vm-1* copies the file "my.data" to *rhel-vm-1*, which is on the LAN. As the NewReno algorithm has excellent performance on the LAN, *aix-vm-1* sends data quickly to *rhel-vm-1*. The socat process on *rhel-vm-1* receives the packet and forwards it to *rhel-vm-2* which is across the WAN. Since the Cubic algorithm has good TCP throughput across the WAN, the packets are sent in a timely manner. Next, *rhel-vm-2* receives the packet and forwards it to *aix-vm-2* on port 22. Note that the final port number is also arbitrary. As another example, the destination could be port 1521 for an Oracle database.

Host *aix-vm-2* believes that *rhel-vm-2* is the client, but the socat process on *rhel-vm-2* receives the return traffic from *aix-vm-2* and forwards it back up the tunnel to *aix-vm-1*. In this way, an end to end socket is created and data flows as expected. The result of this configuration is that data moves approximately 7 to 10 times faster per TCP socket from *aix-vm-1* to *aix-vm-2*.

To reverse the flow (where *aix-vm-2* is the sender), another tunnel would be created that uses a different port than 6000, since 6000 is already in use on *rhel-vm-2*. An example would be to use port 6001. Once the tunnel was setup in the opposite direction, the following command on *aix-vm-2* would use the new tunnel to *aix-vm-1*:

```
scp -P 6001 my.data rhel-vm-2:/tmp/
```

Note that in both cases, the sending host is communicating with a RHEL host on the LAN to send the data. This is important because it determines the TCP throughput performance for the AIX hosts. If *aix-vm-2* were to send data to *rhel-vm-1* directly, then the NewReno TCP congestion control algorithm would be used, and throughput would be limited as before.

4 Production Ready

The socat tunnels, if configured as described in this document, will respawn if the socat process terminates for any reason. While this is a necessary feature, it alone is insufficient for a production environment. A more comprehensive solution has the following attributes:

- High Availability (HA) for the tunnels
- An integrity check on the tunnels (i.e. is the entire tunnel up and passing data?)
- Performance monitoring for the tunnels
- Resource monitoring for the socat processes

The details of a specific configuration are left to the reader as there are too many variants between environments to make specific recommendations. However, **Figure 4** provides a conceptual architecture for a production ready solution:

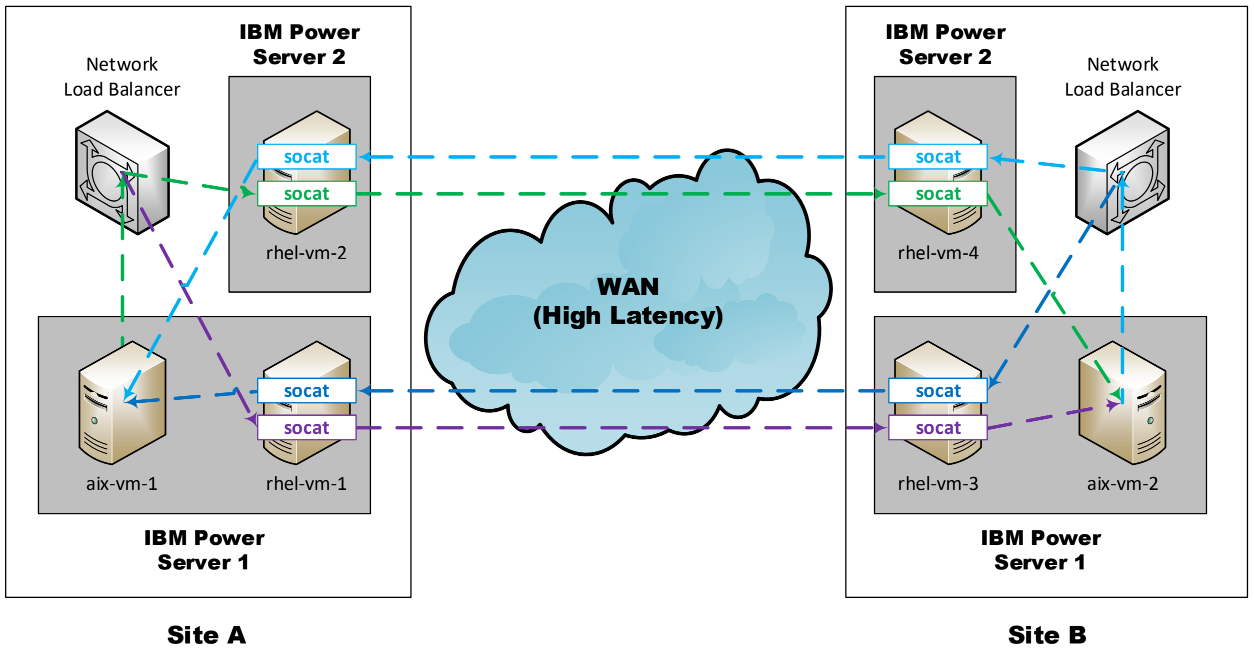

Figure 4: Conceptual Production Architecture

NOTE: Load balancers are only used on the sending side of a connection. The socat tunnels send traffic directly to final host.

Note that the socat processes consume a small amount of resources on the RHEL hosts so be sure they are sized appropriately. The MITRE team observed approximately 10% CPU usage during a large data transfer.

5 Additional Improvements

Keep in mind that the TCP throughput rate limit only applies to a single socket. One way to improve end to end data throughput is to multiplex the transfer across multiple TCP sockets. For example, instead of using four sockets for the transfer, one can use ten sockets. All sockets are operating at the same time so in this example, there would be a 60% increase in data throughput. This increase is independent of the data throughput rate of each TCP socket.

The creation and use of multiple sockets will be determined by the program performing the file transfer. For example, Oracle's recovery manager (RMAN) creates what are known as "channels" to transfer data. Creating multiple channels results in multiple sockets at the TCP layer. The use of multiple sockets combined with the socat tunnel solution in this document should result in significant data throughput improvement between AIX hosts across the WAN.

6 Conclusion

The use of socat tunnels on RHEL hosts to proxy AIX hosts' WAN communications greatly improves the TCP throughput between the AIX hosts. The socat tunnel solution can be implemented now at little to no cost to improve operational data transfers.

Appendix A Scripts and Services

A.1 The socat-tunnel.sh Script

The following is the socat-tunnel.sh script.

Perform these steps on the RHEL host where the socat tunnel will reside:

- Copy the script below to `/usr/sbin/socat-tunnel.sh`
- Execute: `chown root:root /usr/sbin/socat-tunnel.sh`
- Execute: `chmod 740 /usr/sbin/socat-tunnel.sh`

```
#!/bin/sh


Usage() {

     echo "Usage:"

     echo "$0 <local_socat_listening_port> <destination host>:<destination port>"

     exit 1

}


LOCAL_SOCAT_LISTENING_PORT="$1"
```

```
DESTINATION="$2"


# Check parameters

if [ x"$LOCAL_SOCAT_LISTENING_PORT" == x"" ] || [ x"$DESTINATION" == x"" ]; then

     Usage

fi


if [[ "$DESTINATION" != *":"* ]]; then

     Usage

fi


/usr/bin/socat TCP4-LISTEN:${LOCAL_SOCAT_LISTENING_PORT},fork,reuseaddr TCP4:${DESTINATION}
```

Note that the “fork,reuseaddr” options allow multiple TCP connections to the same port. It causes socat to launch a child process for each external host and port that connects to the listening port.

A.2 SOCAT Service File

Be sure that the socat-tunnel.sh script from either **Appendix A-1** or **A-3** is in place before performing this step.

The following is a generic socat-tunnel service definition. Make the following substitutions as indicated:

- X = Listening port on this host
- B = Endpoint host name
- Y = Endpoint host's port number
- b = Next hop host's IP address
- y = Next hop host's port number

Perform these steps on the RHEL host where the socat tunnel will reside:

- For each tunnel that will run, copy the script below to `/usr/lib/systemd/system/socat-tunnel-X-to-B-Y.service`
- Execute: `chown root:root /usr/lib /systemd/system/socat-tunnel-X-to-B-Y.service`
- Execute: `chmod 640 /usr/lib /systemd/system/socat-tunnel-X-to-B-Y.service`
- Execute: `systemctl daemon-reload`
- Execute: `systemctl enable socat-tunnel-X-to-B-Y.service`
- Execute: `systemctl start socat-tunnel-X-to-B-Y.service`
- Execute: `systemctl list-units | grep socat`

```
[Unit]
Description=Start socat tunnel on X to B:Y (service name)
```

```
[Service]

Type=simple

ExecStart=/bin/bash /usr/sbin/socat-tunnel.sh X b:y

Restart=always


[Install]

WantedBy=multi-user.target
```

The “Restart=always” entry causes systemd to respawn the process if it dies. This is important to keep the tunnel up if the process dies unexpectedly.

Note the parameters that are passed to the “socat-tunnel.sh” script. They are the local listening port, the remote host, and the remote port. These will vary for each tunnel service that is created so be sure to substitute the appropriate values as indicated.

A.3 Socat-tunnel.sh Script (Optional)

This section contains an alternative socat-tunnel.sh script. It is identical to the script in **Appendix A.1** except that there is an additional parameter that causes the tunnel use to be restricted to specific source IP addresses. This might be desirable if the SOCAT tunnel should be used only by specific source hosts.

Perform these steps on the RHEL host where the socat tunnel will reside:

- Copy the script below to `/usr/sbin/socat-tunnel.sh`
- Execute: `chown root:root /usr/sbin/socat-tunnel.sh`
- Execute: `chmod 740 /usr/sbin/socat-tunnel.sh`

```
#!/bin/sh


Usage() {

     echo "Usage:"

     echo "$0 <local_socat_listening_port> <destination host>:<destination port>"

     exit 1

}


LOCAL_SOCAT_LISTENING_PORT="$1"

DESTINATION="$2"
```

```
# Check parameters

if [ x"$LOCAL_SOCAT_LISTENING_PORT" == x"" ] || [ x"$DESTINATION" == x"" ]; then

     Usage

fi


if [[ "$DESTINATION" != *":"* ]]; then

     Usage

fi


/usr/bin/socat TCP4-LISTEN:${LOCAL_SOCAT_LISTENING_PORT},fork,reuseaddr,range=10.10.10.20/32
TCP4:${DESTINATION}
```

Note that the “fork,reuseaddr” options allow multiple TCP connections to the same port. It causes socat to launch a child process for each external host and port that connects to the listening port. In addition, the use of the “range” parameter in this version limits the connection to a particular source classless inter-domain routing (CIDR) block. In this case, it is limited to a single host “10.10.10.20”.

Appendix B Abbreviations and Acronyms

The following is a list of abbreviations and acronyms used in this document

Term	Definition
AIX	Advanced Interactive eXecutive
CIDR	Classless Inter-Domain Routing
Cubic	The TCP congestion algorithm used by the Linux TCP/IP stack
GB	Gigabytes
Gbps	Gigabits per second
GBps	Gigabytes per second
IP	Internet Protocol
LAN	Local Area Network
MB	Megabytes
Mbps	Megabits per second
MBps	Megabytes per second
NewReno	The TCP congestion algorithm used by the AIX TCP/IP stack
OS	Operating System
PPC64LE	PowerPC 64-bit Little Endian
RHEL	Red Hat Enterprise Linux
RMAN	Recovery Manager
SCP	Secure Copy
SFTP	Secure FTP
SSH	Secure Shell
TB	Terabytes
TCP	Transmission Control Protocol

VM	Virtual Machine
WAN	Wide Area Network