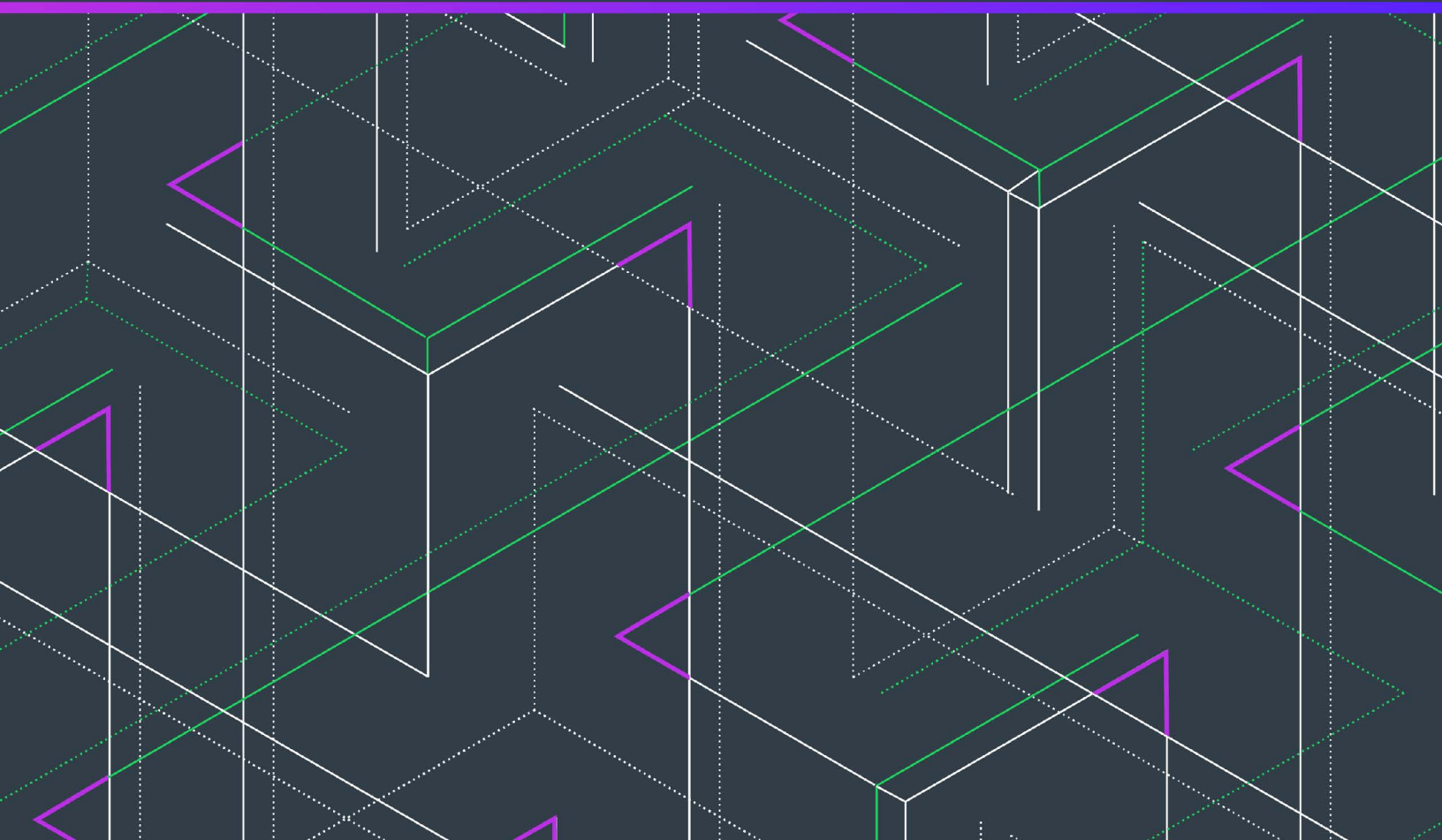


FlexNet Publisher

White Paper: Optimizing FlexNet Publisher License Server Performance



Legal Information

Book Name: FlexNet Publisher White Paper: Optimizing FlexNet Publisher License Server Performance
Part Number: FNP-WP-LSP-1806
Last Updated: August 2021

Copyright Notice

Copyright © 2021 Flexera Software

This publication contains proprietary and confidential information and creative works owned by Flexera Software and its licensors, if any. Any use, copying, publication, distribution, display, modification, or transmission of such publication in whole or in part in any form or by any means without the prior express written permission of Flexera Software is strictly prohibited. Except where expressly provided by Flexera Software in writing, possession of this publication shall not be construed to confer any license or rights under any Flexera Software intellectual property rights, whether by estoppel, implication, or otherwise.

All copies of the technology and related information, if allowed by Flexera Software, must display this notice of copyright and ownership in full.

FlexNet Publisher incorporates software developed by others and redistributed according to license agreements. Copyright notices and licenses for these external libraries are provided in a supplementary document that accompanies this one.

Intellectual Property

For a list of trademarks and patents that are owned by Flexera Software, see <https://www.revenera.com/legal/intellectual-property.html>. All other brand and product names mentioned in Flexera Software products, product documentation, and marketing materials are the trademarks and registered trademarks of their respective owners.

Restricted Rights Legend

The Software is commercial computer software. If the user or licensee of the Software is an agency, department, or other entity of the United States Government, the use, duplication, reproduction, release, modification, disclosure, or transfer of the Software, or any related documentation of any kind, including technical data and manuals, is restricted by a license agreement or by the terms of this Agreement in accordance with Federal Acquisition Regulation 12.212 for civilian purposes and Defense Federal Acquisition Regulation Supplement 227.7202 for military purposes. The Software was developed fully at private expense. All other use is prohibited.

Contents

Understanding FNP License Server Capacity & Performance Measurements	5
A brief history of Traffic theory	5
What do we mean by “FNP License Server Capacity & Performance”	6
Test Environment	7
Test Setup	8
Metrics	8
Recommendations	8
General Recommendations	8
Improve Network Performance	9
Re-use Job Handle	9
Set the Check-In KEEP_FLAG	9
Aggregate Checkouts using LM_A_MULTIPLE_CHECKOUT_DATA	9
Aggregate Checkouts using Batch Checkout	10
Asynchronous lc_vsend()	10
Imstat -no_user_info Option	11
Platform Specific Recommendations	11
AWS Instances	11
Windows Host	11
Non-Windows Hosts	11
Performance Diagnostics	11
Appendix 1	12
Specific Enhancements	12
Appendix 2	12
Windows Connection Limit	12



Optimizing FlexNet Publisher License Server Performance

This white paper suggests strategies for optimizing the performance of the FNP License Server and discusses the recent performance investigations and enhancements that underpin those strategies. The topics covered in this white-paper generally relate to features found in release FNP 11.18.0, although some may have first been provided in an earlier release.

The recommendations are pertinent to typical FNP certificate-based licensing use-cases, and in particular do not consider uses cases relating to Client Trusted Storage, interactions between the License Server and FNO or situations where either client or server is running in a Docker container.

Recommendations given in this White Paper are for general guidance only and are not guaranteed to yield benefits in any particular real-world use-case.

Understanding FNP License Server Capacity & Performance Measurements

A brief history of Traffic theory

Back at the start of the 20th Century telephony was still in its infancy. Subscribers in villages and towns could be connected using copper wire to a local telephone exchange. The exchanges around the country were linked via a network of long distance Trunk connections. When a subscriber wanted to call someone outside of their local area they would need to acquire one of these Trunk connections out of their local exchange in order to reach the local exchange of the called party. In order to minimize the costs associated with installing and maintaining these trunk connections it was important to try to choose the correct number to install for the collection of subscribers connected to a particular local exchange. Too many would see Trunks sitting idle all day. Too few and subscribers would find themselves unable to make calls.

A Danish engineer by the name of Agner Erlang brought some mathematical analysis to the problem, leading to a set of formula called Erlang's equations. These equations relate the incoming telephone-call traffic, in terms of its rate (known as Busy Hour Call Attempts) and the average duration of each call to the probability of the call being rejected either because there were no Trunks available or because the delay acquiring a Trunk exceeded a certain delay. This latter measurement is known as the Grade of Service and can be formally defined thus:-

Grade of Service is the probability of a call in a circuit group being blocked or delayed for more than a specified interval, expressed as a vulgar fraction or decimal fraction.

His contribution was a major factor in the development of telecommunications to the extent that in 1946 the unit of 'traffic' was named after him, the Erlang. The amount of traffic, in Erlangs (E) can be calculated using the following formula:

$$E = \lambda h$$

where λ = call arrival rate and h = average call-holding time.

Erlang's equations have application far beyond telecommunications. Any system in which users compete for a limited resource with a set of performance criteria to determine if the resource has been successfully acquired can be analyzed using Erlang's approach. Erlang was concerned with the Capacity and Performance of a telephone exchange's Trunk connections, we are concerned with Capacity and Performance of the FNP License Server.

What do we mean by “FNP License Server Capacity & Performance”

Capacity

Reducing to simplest terms, the FNP License Server is a repository of licenses each of which can exist in one of two states, checked-in or checked-out.

By default, when we refer to its “Capacity”, we mean its capacity to hold checked-out licenses given a set of constraints. These constraints will include host machine characteristics such as CPU speed, available memory, maximum number of open files and other resource limitations. They may also include imposed performance limits, such as maximum response time to a request.

FNP License Server Capacity is most relevant to use-cases wherein large number of licenses are checked-out from the license server for long periods of time. One could imagine a collection of workstations all equipped with a certain application. They are powered up at the start of the day, run the application throughout the day and are switched off at the end of the day. Minimizing overall startup time for all the workstations is the key requirement, or in other words: Given n checkouted-out licenses, how long does it take to checkout the $n+1$ th license? What is the value of n when that time exceeds a maximum acceptable value?

Performance

Performance is a measure of how fast the FNP License Server can respond to requests to get something done. By default, when we refer to the server's “Performance” we mean the time it takes to check-out a license in a given environment. As with Capacity, the constraints will include host machine characteristics, number of connected clients, the number of checked-out licenses.

Performance is most relevant to use-cases where a license is required as fast as possible and held for a short period of time. A typical scenario might be a high-value licensed compiler that is run at a certain stage of a longer build process. Many engineers may be running builds which need to acquire a compiler license, run the compiler and then release the license. Minimizing the time to acquire a license when the server is handling a heavy load of checkout/checkin requests is the key requirement., or in words, given a continuous checkout/checkin rate of r requests per second, what is the latency l , in seconds for a request to be serviced as seen from the client application.

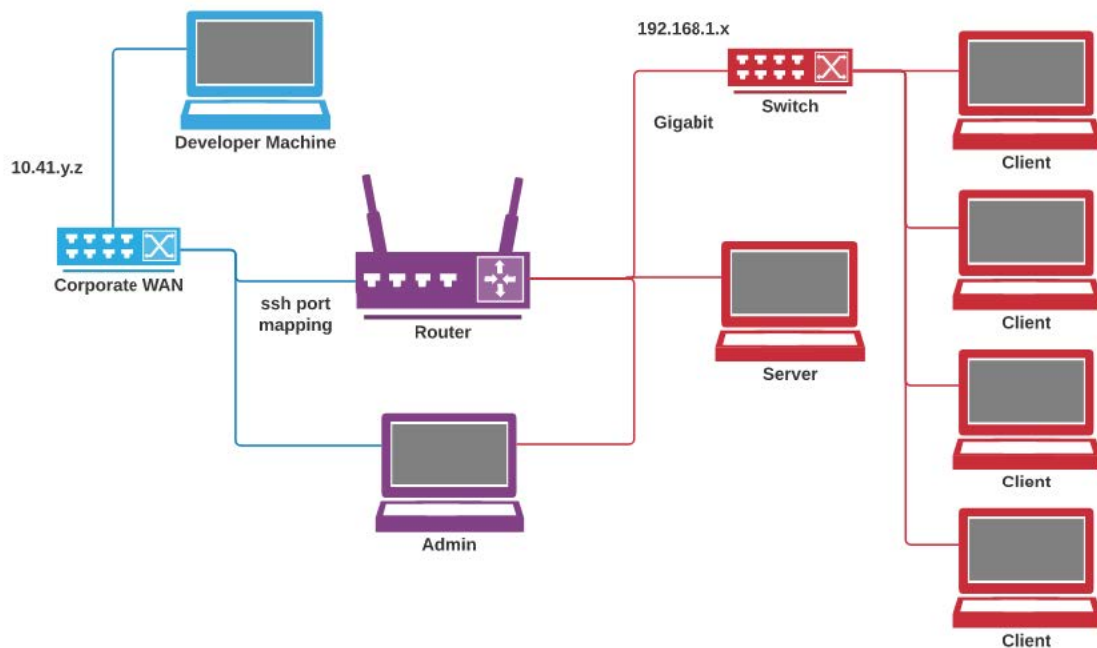
Test Environment

Clearly many factoring will influence Capacity and Performance measurements, some of which will be directly related to the License Server itself and some to the environment in which the measurements are made. In order generate reproducible figures that for both the baseline results and subsequent results after improvements to the license server were made, it was necessary remove as much environmental influence on the test setup as possible. To that end the following arrangement were put in place:

- A dedicated, isolated, ipv4 subnet was create for the test machines.
- All machines had a wired connection to the router running at the same speed.
- A particular machine was always chosen to be the server for a particular test
- A bespoke, configurable, client program (Imperf) was written to generate desired the traffic for each scenario.
- Steps were taken to at random fluctuations to the traffic each Imperf instance generated to avoid spontaneous client synchronization. This can happen when multiple clients has to queue for a resource, they would tend to follow as fixed order of behavior which can skew measurements.

Test Setup

FNP SERVER PERFORMANCE SUBNET



Metrics

The tests performed generally fell into two categories depending on whether the aim was to gather metrics about capacity or performance.

Capacity tests involved gradually incrementing the number of simultaneous checked-out licenses (1 checkout per client connection) in the License Server until the GoS for checking out more licenses fell below a set value. The number of checked out licenses at that point is the measurement of capacity.

Performance tests involved gradually increasing the traffic presented to the server generated by clients running checkout/checkin loops. When the test ended when GoS fell below a set value. The rate at which checkout requests were arriving at the server at that point is the measurement of performance.

Recommendations

General Recommendations

The following sections provide performance guidance that is generally applicable regardless of host platform/environment.

Improve Network Performance

In almost every test-case the most significant factor affecting checkout response times was network latency. This is because a checkout request necessarily involves sequence of message exchanges between client and server. Despite this protocol being optimized in a recent release, improving network latency remains the best way to boost performance.

Re-use Job Handle

The first checkout on for a new job always takes much longer than any subsequent checkouts using the same handle. This is because it is typically the first checkout call which has to establish the connection between client and server, resulting in extra messages being passed and an increase load on the server. Subsequent checkouts will not suffer in these overheads. Note that an improvement was made to reduce the overall impact of establishing new client connections on the server by delegating this activity to a separate thread, this improvement will not affect the latency as seen by the client.

Set the Check-In KEEP_FLAG

Following on from the last section, it should be noted that when the last remaining checkout in a job is checked-in the socket connection to the server is closed. This is a good thing in situations where the client application will not be requiring any further licenses, however in use-cases where the client continually cycles through seizing and releasing all its licenses it creates an unnecessary overhead.

In such use-cases consider setting the 'keep connection' flag in the **lc_checkin()**. This prevents the socket being closed when a job's last remaining licenses is checked-in. Hence the socket connection overhead will be avoided when the next checkout is performed. In this scenario the connection is closed when the job is destroyed.

Aggregate Checkouts using LM_A_MULTIPLE_CHECKOUT_DATA

In use-cases where multiple discreet copies of the same license are required, for example in license proxy use cases, each checkout can be separately labeled by setting the LM_A_CHECKOUT_DATA attribute prior to each checkout.

FPN 11.16.6 added the facility to aggregate all the separate labeled checkout calls into one using the LM_A_MULTIPLE_CHECKOUT_DATA attribute, hence obtaining the licenses in a fraction of the time compared with separate calls.

For example, suppose we require 3 copies of feature f1 to be labeled L1, L2 and L3. Using LM_A_CHECKOUT_DATA we would call **lc_checkout** 3 times, e.g:

```
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, (LM_A_VAL_TYPE)"L1");
lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR)
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, (LM_A_VAL_TYPE)"L2");
lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR)
lc_set_attr(lm_job, LM_A_CHECKOUT_DATA, (LM_A_VAL_TYPE)"L3");
lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR)
```

This could be replaced with:

```
lc_set_attr(lm_job, LM_A_MULTIPLE_CHECKOUT_DATA, (LM_A_VAL_TYPE)"L1\fl2\fl3");  
lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_VENDOR)
```

Note “\f” is the delimiting character.

The end result is identical, expect that the code resulted in just one checkout request being sent to and process by the server.

Aggregate Checkouts using Batch Checkout

Whilst the above technique is useful when requiring multiple discrete copies of the same license, it does not help is use cases where the client application requires multiple different licenses to be checked out from the server.

The Batch Checkout feature provided as an MVP in FNP 11.17.2 and fully released in FNP-11.18.0 offer a mechanism where multiple different licenses can be checked out via a single transaction with the license server rather than 3 separate ones.

For example, Suppose we which to checkout licenses f1,f2, and f3. Historically the code would have looked something like:

```
lc_checkout(lm_job, "f1", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE)  
lc_checkout(lm_job, "f2", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE)  
lc_checkout(lm_job, "f3", "1.0", 1, LM_CO_NOWAIT, &code, LM_DUP_NONE)
```

This can now be achieved with just one interaction with the server using the new Batch Checkout api:

```
lc_txn_create(lm_job, &txn);  
lc_txn_create_checkout(lm_job, &txn, &tx_entry[0], "f1", "1.0", 1,  
    LM_CO_NOWAIT, LM_DUP_NONE);  
lc_txn_create_checkout(lm_job, &txn, &tx_entry[1], "f2", "1.0", 1,  
    LM_CO_NOWAIT, LM_DUP_NONE);  
lc_txn_create_checkout(lm_job, &txn, &tx_entry[2], "f3", "1.0", 1,  
    LM_CO_NOWAIT, LM_DUP_NONE);  
lc_txn_send(lm_job,&txn);
```

One again the net result of the two code snippets is identical, however in the latter example only the final **lc_txn_send()** call resulted in any interaction with the license server, thus reducing the overall execution time to a fraction of that original snippet.

It should be noted that Batch Checkout and LM_MULTIPLE_CHECKOUT_DATA mechanisms are compatible with each other.

Asynchronous lc_vsend()

A long-standing capability offered by the License Server is the execution of a vendor provided function on the license server in response to a client calling the **lc_vsend()** function. Prior to FNP 11.17.2 the execution of the vendor function was handled by the main License Server thread, meaning it could not process any other messages from any other clients until the vendor function completed. This bottleneck has been overcome by devolving the execution of vendor functions to a dedicated thread in the License server, meaning the license server can continue with licensing and other activities in parallel with running vendor functions.

A further vendor function benefit has been provided to clients. Previously a client was blocked in the **lc_vsend()** function until a reply was received from the License Server. It is now possible to set a new attribute, **LM_A_VSEND_NOWAIT**, which causes **lc_vsend()** to return immediately. The result from the License Server can then be polled at some later stage by calling **lc_vsend()** with the **LM_VSEND_STATUS** macro.

Imstat -no_user_info Option

The **Imstat** utility provides a means to obtain detailed license user info from the FNP License Server. In use-cases that involve frequent use of **Imstat**, the burden placed on the license server can become significant and potentially degrade normal licensing operation performance. The impact of **Imstat** calls can be significantly reduced by using the

-no_user_info option, which suppresses detailed per-user information.

Platform Specific Recommendations

AWS Instances

As with the general recommendations above, the limiting factor is most likely to be the network performance between machine instances within the cloud rather than the specification of the instance using for the license server. For example, tests indicate that even using an **r5dn.24xlarge** instance to host the server and with ENA (enhanced networking) enabled the performance could only reach 70% of that achievable on the physical machine using in the Reverera test network which runs a Xeon-2136 processor with 64GB memory. The equivalent AWS instance would be something like a **r5dn.2xlarge**. Going higher than this specification is unlikely to provide any performance benefit.

Windows Host

One additional improvement on Windows platforms is the raising of the maximum number of simultaneous clients a Windows-based License Server can handle from ~30,000 to ~40000. See Appendix 2 for more details.

Non-Windows Hosts

FNP imposes no fixed limit to the number of simultaneous clients that can connect to a non-Windows based License Server, rather it will be OS/Network resource limits or latency constraints that determine a reasonable practical upper limit. In tests Reverera was able to reach 170000 discrete clients using its test network consisting of Xeon-2136 machines.

Performance Diagnostics

A new facility has been added to the License Server to inspect the source and nature of messages being sent to the license server. This can be useful to diagnose whether a particular client or activity is causing excessive load on the License Server and hence allow remedial action to be taken.

The release kit now contains the source for 3 executable programs (log2db.c, log2file.c, log2port.c) and 1 python script (pylog2file.py) that can connect to the license server and report details of all messages arriving at the server.

Appendix 1

Specific Enhancements

The following table details the releases that contain significant performance enhancements:

Table 1 - Performance Enhancement with Release Details

Table A-1 -

Feature / Improvement	FNP Release*	Client Upgrade Required
VM_PLATFORMS protocol simplification	11.16.4 (2019 R2)	No.
VD connection handling thread	11.16.4 (2019 R3)	No.
lmstat -a -no_user_info	11.16.4 (2019 R2)	No, but lmstat upgrade is required.
lc_vsend thread	11.16.4 (2020 R2)	No, unless using client asynchronous option.
FLEX_MSG_TIMEZONE protocol simplification	11.16.4 (2020 R2)	Yes.
Batch Checkout (full release)	11.16.4 (2021 R1)	Yes.

* refers to non-Windows release, typically support on Windows followed later. See relevant Release Notes for specific details.

Appendix 2

Windows Connection Limit

Historically both Windows and non-Windows platforms were constrained in the number of simultaneous client connections, the FNP License Server could handle by their use of the **select()** system call to monitor those connections.

By moving from using the **select()** Operating System call to the **poll()** call, non-Windows platforms were no longer constrained by the limitations of the **select()**, rather the limitation arise from the Operating System and Network configurations. Experiments were undertaken to similarly migrate Windows away from the **select()** call, with

WSAPoll() seeming the obvious choice. Unfortunately it was found that the performance of **WSAPoll** drops off dramatically after around 10k socket connections. Alternative strategies using Microsoft **Overlapped I/O** and **Completion Port I/O** were also explored but found to be unsuitable.

This means that Windows still has to use the old **select()** call and so, whilst the new Connection Thread improves performance somewhat, Revenera suggests that the maximum connection count that can reasonably be supported without excessive latency in the License Server is around 40k simultaneous connections.

