Testing Plan Networked Game:

The main goal is to stress and load test the environment for scaling. And there are 3 main stages where it needs to be done,

1. Game Client (Engine/Gameplay programming)
2. Game Server
3. AWS Infra

We will now go into each stage in depth and see what factors we need to consider for optimizations and are there any tools available or is it possible to have tool for something.
I will consider Unreal Engine and tools around it as the independent study project was developed on UE4.

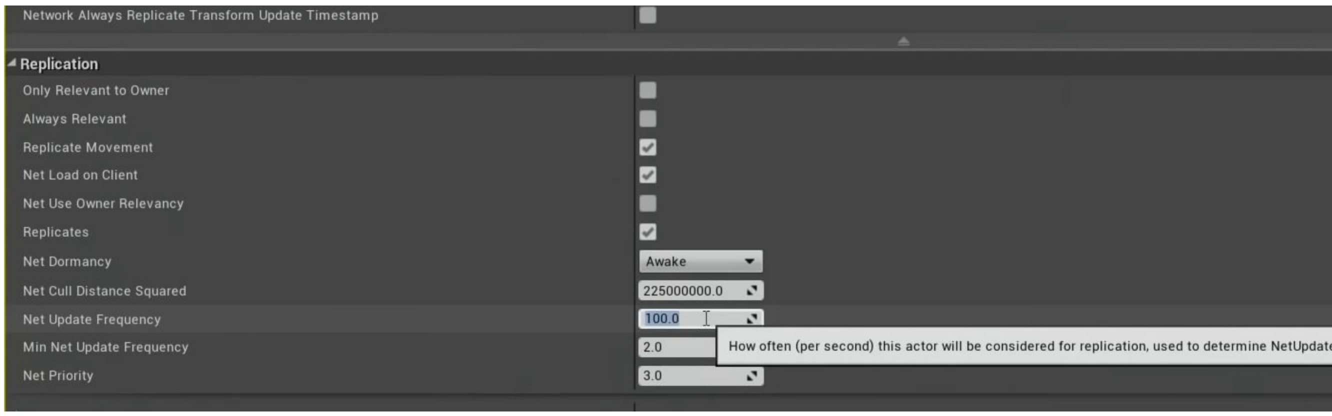
(A) Game Client:

This will consider both engine & gameplay programming aspects but from the client's perspective. We need to consider following things while testing and optimizations for the network performance.

1. **Frequency:**

   You don't need to replicate/update all the objects/actors in the scene all the time. For example, Score, achievement updates to player profile. You need to prioritize what needs to replicated/updated to server every frame and optimize that.

   These kinds of things can be stored locally and can be batched into one server call to reduce requests traffic, whenever you have enough window. Also, you would want to avoid polling at all costs, as suggested in Stop Killing Our Srvers.



Network Always Replicate Transform Update Timestamp	☐
Replication	
Only Relevant to Owner	☐
Always Relevant	☐
Replicate Movement	✓
Net Load on Client	✓
Net Use Owner Relevancy	☐
Replicates	✓
Net Dormancy	Awake
Net Cull Distance Squared	225000000.0
Net Update Frequency	100.0
Min Net Update Frequency	2.0
Net Priority	3.0

How often (per second) this actor will be considered for replication, used to determine NetUpdat

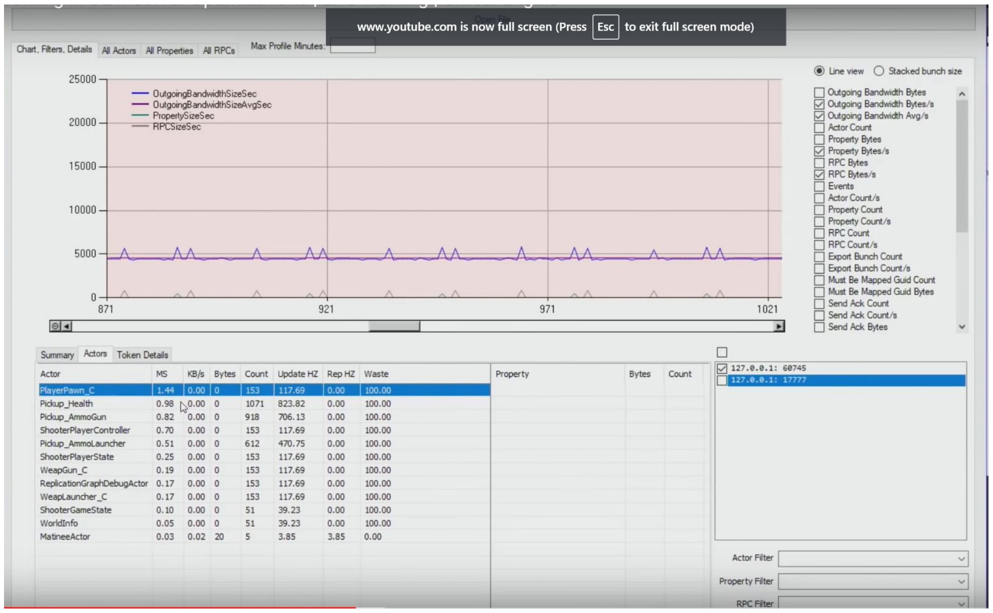
You can play around with these values in the Unreal editor to get better performance. (Performance and Bandwidth Tips)

2. **Reliable and Unreliable calls:**
   Reliable calls a slow and not always necessary. This unreal talk(Networking in UE4) puts emphasis on discussing that.

Tools available for this:

With perspective on UE4, it has a very good networking profiling tool, which is a separate tool than editor which can be used for evaluating network performance. The following figure is an example screen of the tool.

The waste column percentage tells us the % of duration for which data for something is unchanged and we were still replicating it. So our goal is to minimize that.



www.youtube.com is now full screen (Press Esc to exit full screen mode)

[● Line view] Stacked bunch size

- ☐ Outgoing Bandwidth Bytes
- ✓ Outgoing Bandwidth Bytes/s
- ✓ Outgoing Bandwidth Avg/s
- ☐ Actor Count
- ☐ Property Bytes
- ✓ Property Bytes/s
- ☐ RPC Bytes
- ✓ RPC Bytes/s
- ☐ Events
- ☐ Actor Count/s
- ☐ Property Count
- ☐ Property Count/s
- ☐ RPC Count
- ☐ RPC Count/s
- ☐ Export Bunch Count
- ☐ Export Bunch Count/s
- ☐ Must Be Mapped Guid Count
- ☐ Must Be Mapped Guid Bytes
- ☐ Send Ack Count
- ☐ Send Ack Count/s
- ☐ Send Ack Bytes

```
✓ 127.0.0.1: 60745
☐ 127.0.0.1: 17777
```

Actor	MS	KB/s	Bytes	Count	Update HZ	Rep HZ	Waste
PlayerPawn_C	1.44	0.00	0	153	117.69	0.00	100.00
Pickup_Health	0.98	0.00	0	1071	823.82	0.00	100.00
Pickup_AmmoGun	0.82	0.00	0	918	706.13	0.00	100.00
ShooterPlayerController	0.70	0.00	0	153	117.69	0.00	100.00
Pickup_AmmoLauncher	0.51	0.00	0	612	470.75	0.00	100.00
ShooterPlayerState	0.25	0.00	0	153	117.69	0.00	100.00
WeapGun_C	0.19	0.00	0	153	117.69	0.00	100.00
ReplicationGraphDebugActor	0.17	0.00	0	153	117.69	0.00	100.00
WeapLauncher_C	0.17	0.00	0	153	117.69	0.00	100.00
ShooterGameState	0.10	0.00	0	51	39.23	0.00	100.00
WorldInfo	0.05	0.00	0	51	39.23	0.00	100.00
MatineeActor	0.03	0.02	20	5	3.85	3.85	0.00

(B) Game Server:

The game server needs to be tested for two things,

1. **Load Testing -** How many users can the server handle without failing.
2. **Stress Testing -** How far we can push the server for performance

These should help use understand the contentions and bottle necks for the different services we have and based on those we can dig deeper to optimize them,

Tools for this:

(i)A very popular tool for server performance testing is **Apache JMeter**[**(Apache JMeter - Apache JMeter™)**]. You can configure the tool to fire requests to replicate real traffic or just a high number of requests to see where server breaks down.

This is very easily possible to be done in-house with a simple python script too. I had worked on a similar tool during my internship at Veritas LLC. We did load and stress testing with the script and extended it to plot performance graphs with matplotlib.

(ii) If you want to dig deepper into actual code of server functions, and profile the code for server side of engine, the UE network profile tool I mentioned above can be used again.

(C) AWS Infra:

One of the main reasons to use cloud services is the ability to scale up and down automatically based on the traffic.

For gamelift we need to test following things,

1. Fleets Creation
2. If the selected EC2 server is a best fit for game requirement
3. Auto Scaling and descaling

Tools for GameFleet:

(i) We already have a tool to monitor in the gamelift, which can track events happening on the gamelift as it accepts users and scales and descales.

(ii) Testing EC2 is the same thing as serve performance testing (load/stress) so I believe we can use the tools I mentioned above.

We might have other services running as well before we go to actual gameplay,

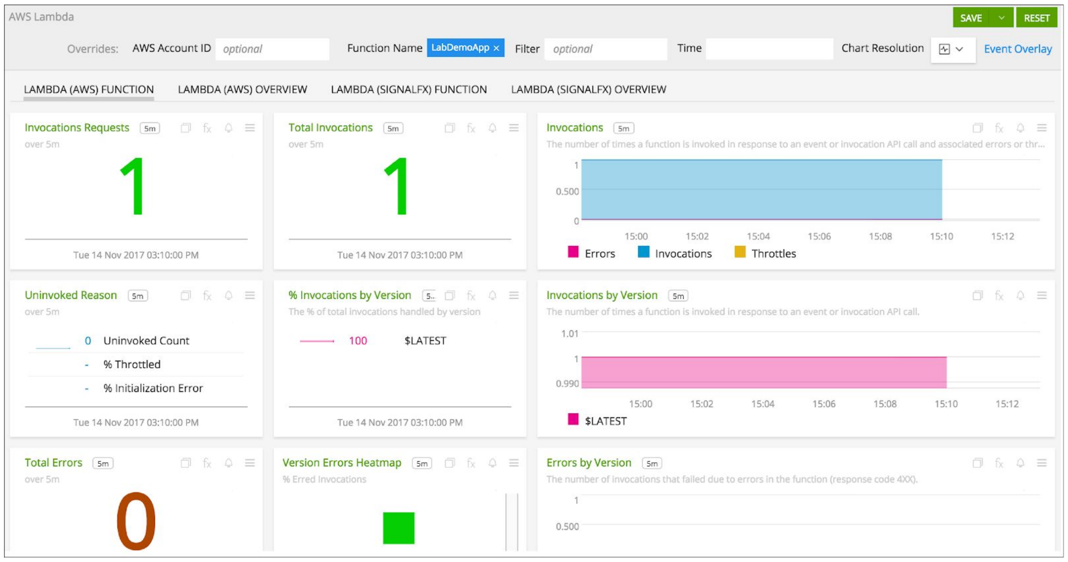
4. Lambda's for signins, data updates, notifications
5. AWS RDS updates for player profiles and other game relevant data

Tools for these:

(i) Lambda's are usually small sets of functions which can do a specific task. For example we can have a lambda to handle steam sign in for user, one lambda to update the player achievements in the db. A lambda needs to be finished in at max 5 min and they run in an async manner. Also By default AWS tries to run a lambda 3 times it fails.

So considering they are used for a specific task and they run in async, I don't really expect them to be a bottleneck.

But It’s a great idea to write a cron job which can fetch a daily report of lambda’s performance. But, AWS console already has a great UI for this.


AWS Lambda
SAVE
RESET
Overrides:
AWS Account ID
optional
Function Name
LabDemoApp ×
Filter
optional
Time
Chart Resolution
Event Overlay
LAMBDA (AWS) FUNCTION
LAMBDA (AWS) OVERVIEW
LAMBDA (SIGNALFX) FUNCTION
LAMBDA (SIGNALFX) OVERVIEW
Invocations Requests 5m
over 5m
1
Tue 14 Nov 2017 03:10:00 PM
Total Invocations 5m
over 5m
1
Tue 14 Nov 2017 03:10:00 PM
Invocations 5m
The number of times a function is invoked in response to an event or invocation API call and associated errors or thr...
Errors
Invocations
Throttles
Uninvoked Reason 5m
over 5m
0 Uninvoked Count
- % Throttled
- % Initialization Error
Tue 14 Nov 2017 03:10:00 PM
% Invocations by Version
The % of total invocations handled by version
100 $LATEST
Tue 14 Nov 2017 03:10:00 PM
Invocations by Version 5m
The number of times a function is invoked in response to an event or invocation API call.
$LATEST
Total Errors 5m
over 5m
0
Version Errors Heatmap 5m
% Erred Invocations
Errors by Version 5m
The number of invocations that failed due to errors in the function (response code 4XX).

(ii) Since we are going for microservices architecture, we should test that if failing of one service is not restricting user from having any possible experience at all. We could test by disabling a few lambdas and firing service requests.

This is the basic test plan with which I’d approach to test a game for network performance.