Actor Network Dormancy

Optimize your multiplayer game by effectively using dormancy.

Actor **Network Dormancy** (`AActor::NetDormancy`) is one of the most impactful server optimizations a multiplayer project can make, potentially saving multiple milliseconds of server CPU time per frame — especially if your project has a large number of replicated actors that don't change often. During a net update, the NetDriver gathers a list of all the replicated actors relevant to a connection, then iterates over these actors and their properties to determine what has changed and should be sent to the client. An actor's network dormancy controls whether the actor is added to the connection's gathered actors list. *Dormant* actors are not added to the list while *awake* (not dormant) actors are added to the list. Iterating over actors and their properties can be expensive, especially for projects with a large number of replicated actors. As a result of this, effectively using network dormancy to filter actors out of the considered for replication list can be a vital optimization for multiplayer projects.

How to Use Network Dormancy

Follow these steps to use network dormancy effectively in your project:

1. Set the actor's `AActor::NetDormancy` to `ENetDormancy::DORM_DormantAll` in its constructor:

```
    NetDormancy = ENetDormancy::DORM_DormantAll;
```

Copy full snippet

2. Set the actor's `AActor::NetDormancy` to `ENetDormancy::DORM_Initial` if the actor is placed in the map.
3. While dormant, the actor will not replicate.
4. To replicate the actor, call `AActor::FlushNetDormancy`, `AActor::ForceNetUpdate` or `AActor::SetNetDormancy(ENetDormancy::DORM_Awake)` before changing any replicated properties.
5. For replicated components or subobjects, the owning actor must be flushed or woken up before their replicated properties can change.

When a replicated actor is marked as dormant, the `NetDriver` is able to skip adding it to the connection's gathered actors list. This saves time otherwise spent considering the actor for replication and comparing its properties. Until a dormant actor is awoken or has its dormancy flushed, the actor is not considered for replication; therefore, any changes to its replicated properties are not sent to clients. It is also important to note that an actor's replicated state should not change while it is dormant, as these changes might be lost when the actor is awoken. This means that actors that change their replicated properties infrequently are good candidates for dormancy. Actors that require frequent updates, such as pawns, are not likely to benefit, and, due to the additional overhead of changing or flushing an actor's dormancy state, doing this too frequently might be even less performant than just keeping the actor awake.

While the actor channel for a replicated actor will close when it goes dormant, dormant actors still exist on both the server and client. This is different to how Actor Relevancy is handled: dynamic, replicated actors are destroyed on the client when they are no longer relevant. It is worth noting that dormant actors are not checked for relevancy, so if a dormant actor would otherwise go out of relevancy on a client, it is not destroyed on that client (unless you are using a Replication Graph with the `Net.RepGraph.DormantDynamicActorsDestruction` console variable enabled).

Change an Actor's Network Dormancy

An actor's dormancy state is stored in its `AActor::NetDormancy` property. This property can be changed by calling `AActor::SetNetDormancy`. While `AActor::NetDormancy` is public, this property should not be set directly outside of the actor's constructor. `AActor::SetNetDormancy` should be called to change an actor's dormancy, as this function also handles notifying the `NetDriver` of the change.

Network Dormancy States

There are five dormancy states described in the `ENetDormancy` enumeration class (defined in `EngineTypes.h`). The table below describes these dormancy states:

Network Dormancy State	Description
`DORM_Never`	This actor never goes dormant. This is not enforced by the dormancy system. An actor marked as `ENetDormancy::DORM_Never` can be made dormant.
`DORM_Awake`	This actor is not dormant and is considered for replication.
`DORM_DormantPartial`	This actor is dormant on some connections, but not all. Use `AActor::GetNetDormancy` to determine which connections the actor is dormant for. We do not recommend that you use partial dormancy. Partial dormancy is planned for deprecation.

Network Dormancy State	Description
`DORM_Initial`	This actor is initially dormant on all connections. This should only be used by actors initially placed in the map.
`DORM_DormantAll`	This actor is dormant on all connections.

Wake a Dormant Actor

There are two primary ways to wake a dormant actor:

- Set the Net Dormancy Property.
- Call the Flush Net Dormancy Function.

Set Net Dormancy

You can wake a dormant actor by calling `AActor::SetNetDormancy(ENetDormancy::DORM_Awake)`. While the actor is awake, it replicates as normal until it is set back to dormant by calling `AActor::SetNetDormancy(ENetDormancy::DORM_DormantAll)` from the actor. This is useful in cases where a dormant actor is going to begin changing every frame, such as a stationary object beginning to move.

Once a statically placed, initially dormant actor has been awoken, you should not set it back to `ENetDormancy::DORM_Initial`. Use `ENetDormancy::DORM_DormantAll` instead.

Flush Net Dormancy

You can also replicate changes for a dormant actor by calling `AActor::FlushNetDormancy` on the actor. This forces the actor to replicate at least one update to all connections on which it is relevant without actually changing its dormancy state. There is one exception to this, if you call `AActor::FlushNetDormancy` on an actor that has its `AActor::NetDormancy` set to `ENetDormancy::DORM_Initial`, the call to `AActor::FlushNetDormancy` changes the actor's dormancy to `ENetDormancy::DORM_DormantAll`.

Force Net Update

Calling `AActor::ForceNetUpdate` on a dormant actor will also call `AActor::FlushNetDormancy`, while also ensuring the actor is considered for replication on the next net update. This is useful for infrequent, one-off updates to an actor that happen in a single frame.

After an actor's dormancy is flushed (or when it is set dormant after being awake), the actor may send multiple updates as it does not immediately become dormant. Instead, the actor continues to replicate until it and its subobjects have no more unacknowledged changes that should be sent. If dormancy hysteresis is enabled, it also keeps the actor from immediately going dormant (see `UActorChannel::ReadyForDormancy` and `FObjectReplicator::ReadyForDormancy`).

When to Use Wake Methods

`AActor::SetNetDormancy(ENetDormancy::DORM_Awake)` or `AActor::FlushNetDormancy` should be called on an actor before making changes to its replicated properties. Waking a dormant actor reinitializes the actor's *shadow state* (the state used to compare what properties have changed and need to be replicated) by copying all the current values for its replicated properties. This is also why an actor's replicated state shouldn't be changed while it is dormant, as when the actor is awoken, these changes won't be detected when comparing its properties for replication.

In most cases, however, calling it after making changes might still result in those changes being replicated as expected. However, this behavior is an implementation detail, and you shouldn't rely on it. For instance, there are certain situations, such as when changing a dormant fast array, where calling `AActor::FlushNetDormancy` after changing the property results in the changes not being replicated at all.

Dormancy with Blueprint Actors

When using dormancy with a Blueprint actor, the actor automatically calls `AActor::FlushNetDormancy` when you set a replicated property.

> This does not happen automatically when setting replicated properties on an `ActorComponent` blueprint, but we are working to make this behavior consistent.

Dormancy and Replication Graph

When using the Replication Graph, dormancy should operate the same as when using the default `NetDriver`, so project code can set actors as dormant/awake and call `AActor::FlushNetDormancy` as normal. Even if a node returns a dormant actor when gathering its actor list, `UReplicationGraph::ReplicateActorListsForConnections_Default` still skips replicating these actors.

Replication Graph nodes can include special handling for dormant actors. This can reduce the time and memory spent processing dormant actors for a node, as well as reduce the size of the node's gathered actors list. For example, the `GridSpatialization2D` node includes extra handling for dormant actors, treating them as static when they are dormant and dynamic when they are awake. This can be useful for actors that may sometimes move throughout the grid but can also remain stationary and dormant at other times.

Debugging Network Dormancy

Logging

Enable the `LogNetDormancy` log category to get information in your logs on the dormancy status of actors. Increasing the verbosity of this category logs more detailed information, such as when dormancy is flushed for any actor.

Console Variables

The table below contains several console variables related to network dormancy that you might find helpful:

Console Variable	Description
`net.DormancyEnable`	Use to fully enable or disable dormancy for all actors. This is useful for determining if a replication issue is related to dormancy or not.
`net.ReuseReplicatorsForDormantObjects`	If this is enabled, the server attempts to reuse a dormant object's `FObjectReplicator` rather than destroying them when the actor goes dormant and recreating them when the object awakens. ❗ This functionality is no longer used internally. Keeping these replicators around increases server memory usage. For projects with a large number of dormant actors, this additional memory usage can be substantial. Reusing these replicators can also result in issues when replicating actors in sublevels.
`net.DormancyValidate`	If this is enabled, print warnings to the log if actors are changing their replicated properties while dormant. • Set to 1 to validate dormant actors only when they are awoken. • Set to 2 to validate dormant actors every net update. Enabling dormancy validation causes dormant objects' replicators to be reused, as these are needed to compare the replicated properties while dormant.
`net.DormancyHysteresis`	The time (in seconds) the actor channel waits before becoming fully dormant. This is set to 0 by default, but increasing it might prevent churn in cases where actors are rapidly going in and out of dormancy.

Console Variable	Description
`Net.RepGraph.LogNetDormancyDetails`	If this is enabled, more detailed log info is printed on how the replication graph is handling dormant actors.
`Net.RepGraph.DormantDynamicActorsDestruction`	If this is enabled, dormant actors are destroyed on the client when they go out of relevancy. See the Console Variables at the top of `ReplicationGraph.cpp` for more ways to configure this behavior.

For more information about console variables in Unreal Engine, see the Console Variables documentation page.