

ScarNet

Multiplayer Automation & Debugger

BLUEPRINT DEVELOPER GUIDE

Make any actor multiplayer-ready in one click, replicate variables and fire network events without touching an RPC, and find out *why* your netcode is broken with the in-game ScarNet Debugger.

Unreal Engine 5.2 to 5.8 • Blueprint-first, C++ compatible • Works with Dedicated / Listen / LAN / Steam / EOS

Black Scar Studio • Version 1.3.1

1 What ScarNet does

Multiplayer in Unreal is 10% gameplay and 90% “why isn’t this replicating.” ScarNet does three things about that:

- **Automates the boring, error-prone setup.** Add one component, pick a profile, and your actor is replicated with sane update rates, priority, dormancy, relevancy and ownership. Replicate variables and fire network events straight from Blueprint.
- **Tells you what is broken and how to fix it.** Press F9 in game for a live debugger. Its **Issues** tab names each problem in plain English, one line each, with a one-line fix.
- **Checks your whole game, not only ScarNet actors.** It reads Unreal’s own replication records for every replicated actor, counts every RPC your game sends, and turns the warnings Unreal’s networking code prints into readable issues.

It does not replace Unreal networking. It sits on top of the engine’s replication layer, so it works with any Online Subsystem and any server type with no backend-specific code.

TIP The golden rule of this plugin

When something doesn’t sync, don’t guess. Press **F9**, read the top red line on the Issues tab, and do what it says. That loop is the whole point of ScarNet.

2 Install & enable

- Copy the **ScarNet** folder into your project’s `Plugins/` directory.
- Open the project. If prompted to rebuild modules, click **Yes**.
- Confirm it is on under **Edit** → **Plugins** → **Networking** → **ScarNet** (it is enabled by default).
- Tune anything you like under **Project Settings** → **Plugins** → **ScarNet**.

TIP C++ not required

Everything in this guide is done with Blueprint nodes and the Details panel. The plugin is fully C++ compatible too, but you never have to open a .cpp file.

3 Quick start in 4 steps

#	Do this	Result
1	Open your actor Blueprint. Add Component → ScarNet Ready .	The actor can now be made multiplayer-ready.
2	In the component Details, pick a Replication Profile (Character, Weapon, Projectile, Pickup, and so on).	On Begin Play the actor is replicated with tuned settings for that kind of actor.

#	Do this	Result
3	Use the ScarNet nodes: Register / Set Replicated Variable, Broadcast / Send To Server / Send To Owner, MP Spawn / Interact.	Variables sync and events fire on the right machines automatically.
4	Play with 2+ players and press F9 .	The ScarNet Debugger opens and shows any problems with fixes.

WATCH OUT Call setup on the server, at runtime

Add the component, apply profiles and register variables from **Begin Play** (or later) on the **server**, not in a Construction Script and not on a client. Components added on a client don't replicate. ScarNet warns you in the log if you get this wrong.

4 Replication profiles

A profile is a curated bundle of networking settings for a *kind* of actor. Pick the closest one and ScarNet applies proven defaults; choose **Custom** to set them by hand. All values are editable per project in **Project Settings** → **ScarNet**.

Profile	Use it for	What it tunes
Character	Player pawns	High update rate, replicated movement, always awake
Projectile	Bullets, rockets, grenades	High rate and replicated movement while alive
Weapon	Equipped weapons	Medium rate, relevant through the owner
Interactable	Doors, buttons, levers, switches	Dormant until used, then flushed
Pickup	World pickups	Low rate, dormant until grabbed
Inventory	Private per-player state	Only relevant to the owning player
AI	AI pawns	Medium rate, replicated movement
Custom	Anything else	You set every value on the component

TIP Movement replication is on by default

The ScarNet Ready component forces **Replicate Movement** on (and fixes a Static root that would silently block it). Moving actors just work. Turn it off per component only for truly static objects.

5 The Blueprint nodes

All nodes handle authority checks and server routing for you. They work on any machine and any net mode. Call them where it reads naturally and ScarNet does the right thing.

Setup

Node	What it does
Make Multiplayer Ready	Adds a ScarNet Ready component (if missing) and applies a profile. Great for actors you spawn at runtime.

Replicated variables

Node	What it does
Register Replicated Variable	Creates a synced variable with a default value. Replicates to everyone, including late joiners.
Set Replicated Variable	Writes it. Automatically routed to the server when called on a client. Fires the change event.
Get Replicated Variable	Reads the local copy anywhere.
Unregister Replicated Variable	Removes it everywhere (server only).

To react to a change, select the **ScarNet Ready** component and bind **On Replicated Variable Changed**. It fires on the server and every client. Use the **As Bool / As Int / As Float / As Vector** nodes to unpack the value.

Network events

Node	Runs on
Send To Server	The server only (a client asks the server to do something). Works from any actor.
Broadcast Event	The server and every client (explosions, round start, global effects).
Send To Owner	The client that owns the target actor (ammo, private messages).
Network Event	Any of the above. You choose the scope on a dropdown, including Run Locally.

Receive events by binding **On Network Event** on the target's ScarNet Ready component, then branch on the event name.

Smart actors & interaction

Node	What it does
MP Spawn Actor	Spawns on the server from anywhere and replicates. On the server it returns the actor; on a client it returns None and the actor arrives through replication a moment later.
MP Destroy Actor	Destroys on the server from anywhere (client requests obey the security settings).
MP Interact	Interacts with an actor that has a ScarNet Interactable component. Works from any machine and is validated (distance-checked) on the server.

TIP MP Spawn returns None on clients, and that is correct

The actor is created on the **server** and replicated down; it doesn't exist on the calling client yet. React to the actor in its own Begin Play instead of using the return value on a client.

6 Auto variables & the “my RepNotify never fires” trap

ScarNet only sends a variable when its value actually **changes** (this saves bandwidth). That also means: if you set a variable to the value it already has, nothing replicates and the change event does **not** fire. This is the single most common multiplayer head-scratcher, so ScarNet makes it visible.

- The **Variables** tab shows a **No-Change Sets** counter climbing.
- The **Issues** tab warns, for example: *‘Health’ set 12 times to the same value: nothing sent, change event never fires.*
- If you genuinely need the callback to fire anyway, tick **Force Notify** on Set Replicated Variable.

WATCH OUT If a client callback “never runs”

Most of the time the value simply isn't changing on the server. Check the No-Change Sets column before suspecting replication itself.

7 The ScarNet Debugger (F9)

Press **F9** in any play session (or run the `ScarNet.Debugger` console command). It opens straight to a status banner and the Issues tab: the answer, not a wall of data. Drag the header to move it and the corner to resize it.

Read the banner first

Banner	Means	Do
AMBER	You are in Standalone . No networking is running.	Set Number of Players to 2+ (or Net Mode: Play As Client) in the Play dropdown.
RED	Problems were found.	Open the Issues tab (it is already in front) and read the top line.
GREEN	No replication problems detected.	Carry on. Move and interact; issues appear the moment they happen.

The five tabs

Tab	Shows
Issues	Every detected problem, most severe first, each with a one-line fix. Start here.
Actors	Every replicated actor: a State column (OK, INFO, WARN, ERR) driven by the Issues list, then Replicates, Movement, Activity (really moving?), Last Sent, owner, authority, dormancy, update rate and relevancy.
Variables	Every registered variable: value, updates, and No-Change Sets.
Events	First, your game's RPCs, one row per function: how many calls, calls per second, and Reliable or Unreliable. Calls Unreal threw away show in red. Below them, the ScarNet event history with Sent, Received, Failed and Dropped.
Health	Ping, packet loss, bandwidth, connections, active and dormant actors, saturation. On a client, Replicated Actors shows how many the server sends you.

TIP Debug from any client

Many problems are only visible on the server. When you open the debugger on a **client**, ScarNet streams the server's findings down to you, so both sides show the same problems. Unreal gives a spawned actor a different name on each machine, so ScarNet shows every actor by the name *this* machine uses. Text is clipped to fit: **hover any cell** to read the full line.

Bottom bar: **Ownership Overlay** draws boxes in the world (green = server, blue = client, yellow = shared, red = problem). **Save Report** writes a full diagnostics text file to `Saved/ScarNet/` (the on-screen message shows the exact path). **Clear Events** wipes the RPC history and the engine warnings for this session, so you can retest from a clean slate.

8 How ScarNet decides

A debugger that cries wolf is worse than none. ScarNet only reports what it can back up, and the severity tells you how serious it is.

Severity	Means	Examples
ERROR	Broken right now. Players see it.	An actor frozen on clients. A Server RPC Unreal threw away.
WARNING	Works for now, but costs you or breaks under load.	Reliable RPC spam. A hitch while an asset loads mid-game. Clients drifting out of sync.
INFO	A tip to save bandwidth, or something ScarNet is still confirming.	Idle actors that could sleep. A moving actor no client has confirmed yet.

Moving actors: the clients get a vote

When the server moves an actor that is not sending its movement (Replicate Movement off, or Dormant), that is only a bug if clients do not see it move. Many games move hazards, platforms and saw blades on every machine from a small replicated state, and that is correct. So ScarNet asks the connected clients to compare what they see with the server's position. It allows for the time the check takes to arrive and for each client's ping, and it only reports what a client confirms twice in a row. One-off moves count too: a server teleport that clients never receive is caught the same way.

Clients see the actor	ScarNet shows
Moving, in the same place as the server	Nothing. Your setup is correct.
Standing still, or stuck where it was before a teleport	ERROR Frozen on clients, with the reason (Replicate Movement off, or Dormant).
Moving, but somewhere else	WARNING Out of sync, with the distance in meters (after allowing for ping).
No client has answered yet	INFO Moved on the server (not yet confirmed by a client). Many of one class become a single line.

TIP Example: the door that never opens

A door uses Dormant All and opens without **Flush Net Dormancy**. The server sees it swing open, both clients report it never moved, and the Issues tab shows: *ERROR Door_3: Frozen on clients: moved on the server, but it is Dormant*. Add Flush Net Dormancy before moving it and the line disappears.

An actor set to Dormant All that never stops moving never actually goes dormant, so clients do receive its movement. The clients confirm this and ScarNet stays quiet.

TIP Example: saw blades that chase different players

In a real test game, saw blades moved on every machine and each machine chose its own nearest player to chase. Most of the time the copies matched. When a client picked a different player, it saw the blade 44 m from where the server had it, and ScarNet reported *WARNING BP_SawBlade_C_4: Out of sync: clients see it at least 43.8 m from the server's position*. The fix is to replicate the target (or the movement) so every machine agrees.

Variables: the clients compare them too

Every few seconds the server sends a short fingerprint of a few actors' variables to each client, and each client compares it with its own copy. The same rule as movement applies: a difference only counts when it holds twice in a row while neither side is changing the value, so a value that is still on its way is never reported.

What a client has	ScarNet shows
A different value in a variable that is not replicated, which the server changed	INFO 2 variable(s) change on the server but never reach clients (<i>not replicated</i>): <i>Score, Ammo</i> . Fine if only the server uses them.
A different value in a replicated variable	WARNING a client changed it locally. Only the server should set it.
The old value, on an actor that is Dormant	WARNING changed while Dormant. Call Flush Net Dormancy first.
No value, because of a Replication Condition such as Owner Only	INFO only reaches some players. Expected if you meant it.
A different Owner	WARNING Set Owner was called on a client, which the server ignores.
A different attach parent	WARNING attached or detached on one side only.

Simple variables are compared (true or false, numbers, enums, names, text, vectors, rotators and colors), up to 24 per class and 16 actors per check, so the cost stays tiny. Rotators are compared the way Unreal sends them (16 bits per axis), so rounding never looks like a difference.

RPCs that run on the wrong machine

Some RPC mistakes make Unreal run the call on the wrong machine without any warning. ScarNet watches C++ RPC calls as they run (Development and Debug builds) and reports them as errors:

- **Multicast called on a client:** it runs only on that client. Other players never see it.
- **Run on Server that only ran on the client:** called on an actor that exists only on that client and has no owner, so the server never hears about it.
- **Run on owning Client on an actor nobody owns:** it runs on the server instead of on a player.

Blueprint graphs call RPC events inside the Blueprint VM, where this hook cannot see them. The Project Check (below) covers Blueprints by reading the graphs instead.

Setup that can never replicate

- **Replicated variables on an actor or component with replication off:** those variables never reach clients.
- **UPROPERTY(Replicated) without its DOREPLIFETIME line (C++):** it never replicates. ScarNet names the variable.
- **A replicated pointer to something clients can never receive:** an actor spawned on the server that does not replicate, or a UObject that is not a replicated subobject. Clients get None.
- **A replicated actor spawned on a client:** nobody else ever sees it.
- **A moving actor sending movement fewer than 10 times a second:** clients see it jump.

Short-lived actors are remembered

A projectile or hazard often lives a few seconds. If clients confirmed a problem on it before it was destroyed, the problem stays in the list for its class: *WARNING BP_SawBlade_C: Seen on 3 destroyed instances: Out of sync: clients see it at least 42.1 m from the server's position*. Clear Events resets it.

Unreal's own warnings

Unreal's networking code already prints warnings to the log that most developers never see. ScarNet turns them into issues with a fix, on the machine where they happen:

- **No owning connection:** a client called a Server RPC on an actor it does not own. The call never reaches the server.
- **RemoteRole is ROLE_None:** a client called a Server RPC on an actor that only exists on that client.
- **Reliable buffer overflow:** too many reliable RPCs were queued and the player was disconnected.
- **Failed validation:** a client sent values a With Validation check rejected, and that player was disconnected.
- **Access rights:** an RPC was sent by the side that is not allowed to send it.
- **Not supported for networking:** a replicated reference points to an object Unreal cannot send.
- **Long net serialize:** receiving a reference to an object took a long time, usually a hitch from loading an asset mid-game.
- Anything else from Unreal's networking log is shown as it was written, with its count.

Your RPC traffic

While the debugger is open, ScarNet counts every RPC your game sends, per function. It warns when one Reliable RPC is sent more than 30 times a second (it can overflow and kick the player), and it marks calls Unreal will throw away because no player owns the actor. This counting uses an engine hook that exists in Development and Debug builds, not in Shipping builds. Everything else works in Shipping.

Optimization tips

- **Idle actors:** 106 replicate at 100 Hz but sent no updates for 30+ s. Unreal still checks every one of them on every update. Make them Dormant and call Flush Net Dormancy when they change.
- **Net Update Frequency 0:** the actor sends once and then never updates, and players who join later may never receive it.
- **Dormancy Initial on a spawned actor:** behaves like Dormant All. Use Dormant All on purpose and flush on change.
- **Too many Always Relevant actors:** each one goes to every player on every update. Prefer distance relevancy.

9 Check your project without playing

Tools → **ScarNet** → **Check Project for Network Mistakes** reads every Blueprint in `/Game` and the C++ bodies of your game's RPC functions, and lists mistakes in the **ScarNet** message log. Click a Blueprint line to jump to the node. Nothing has to run, so it also catches code your test session never reached.

TIP Example: the explosion only one player sees

An input event calls the Multicast *Multi_Boom* directly. On the host it works; on every client only that player sees the explosion. The check reports: *ERROR BP_Grenade (EventGraph > Multi Boom): Multicast 'Multi_Boom' is called from an input event: it only runs on that client.* Call a Run on Server event from the input, and call the Multicast inside it.

It follows each chain of nodes from where it starts (input events, widgets, the HUD, both pins of Switch Has Authority, replicated events, Event Possessed, Event Tick) into your own functions and collapsed graphs:

The graph (or C++ RPC body) does	Why it breaks
Calls a Multicast on a client	It only runs on that client.
Sets a replicated variable on a client, and nothing there tells the server	The change stays on that client.
Spawns a replicated actor on a client	Other players never see it.
Spawns a replicated actor inside a Multicast	The server's copy reaches everyone and each client spawns its own: duplicates.
Calls Destroy Actor on a client	A client cannot destroy a replicated actor. Nothing happens.
Calls a Reliable RPC from Event Tick	Sent every frame, it can overflow and disconnect the player.
Uses Get Game Mode on a client	The GameMode exists only on the server: None.

The graph (or C++ RPC body) does	Why it breaks
Uses Get Player Controller (or GetFirstPlayerController) in server code	Returns the host or first player, not the one who called.
Creates a widget in server code (Run on Server, Authority pin, Event Possessed)	Widgets exist only on clients.
Plays a sound or effect only inside a Run on Server event	Only the server sees or hears it (shown as INFO).
Has replicated variables or events in a widget, the HUD or the GameMode	They never cross the network.
Has replicated variables on an actor with Replicates off, or adds a component with Component Replicates off	They never reach clients.

Client prediction is not reported: changing something locally **and** telling the server in the same chain is a normal technique. A base class that leaves Replicates off is not reported when a child class turns it on, and a component is judged where an actor adds it.

```

Console:          ScarNet.CheckProject
Build machine:    UnrealEditor-Cmd.exe MyGame.uproject -run=ScarNetCheckProject
                  (add -Path=/Game/Folder to check one folder of Blueprints)
Report:           Saved/ScarNet/NetworkCheck.txt (exit code 1 when errors are found)

```

10 How much it catches, and what it cannot

We collected 52 replication mistakes that Unreal developers commonly hit and checked each one against ScarNet, separately for C++ and Blueprint:

	Mistakes that apply	Caught	Of those, planted and tested
C++	51	41 (80%)	35
Blueprint	49	41 (84%)	18

What it cannot catch, so you know when to look elsewhere and never trust a green banner too far:

- **Intent:** an Unreliable RPC used for an event that must arrive, a Multicast used for state that late joiners need, Has Authority where Is Locally Controlled was meant, a C++ OnRep expected to run on the server. The code is valid; only you know what it was meant to do.
- **Character movement corrections** (rubber-banding when speed or movement mode is changed on one side only). Unreal does not expose a count of them.
- **State lost across seamless travel.**
- **RPC calls made from Blueprint graphs at run time** are invisible to the runtime hook. The Project Check reads the graphs instead, which covers the common cases.
- **Movement checks need a connected client.** A listen server host playing alone can only show “not yet confirmed”.

- **Very fast movers on a laggy connection:** the ping allowance can hide a small drift. The distance shown is always a lower bound.
- **Shipping builds** have no per-function RPC list and no RPC hooks, because the engine hooks they use do not exist there. Test in Development builds.
- **Gameplay logic.** A wrong damage formula is not a replication bug.

11 Troubleshooting: symptom → fix

The Issues tab detects all of these automatically. This table is the same knowledge for offline reference.

Symptom	Likely cause	Fix
Actor is frozen on clients	The server moves it but Replicate Movement is off, or it went dormant and moved without a flush.	Enable Replicate Movement, or call Flush Net Dormancy when it moves.
Clients see it in the wrong place	Each machine moves it itself and they drift apart.	Replicate Movement, or drive the motion from replicated state and Get Server World Time Seconds.
A Server RPC called from a client never runs	The client does not own the actor (no owning connection).	Call it on the Player Controller or Pawn, set the actor's Owner on the server, or use Send To Server.
A Server RPC on an actor the client spawned never runs	That actor only exists on the client.	Spawn it on the server with MP Spawn Actor.
Player disconnects while firing fast	A Reliable RPC is called too often and the buffer overflows.	Make frequent calls Unreliable, or send less often.
A hitch when something first appears	An asset loads mid-game when its reference arrives.	Load it earlier (hard reference or preload).
Nothing replicates at all; only 1 window	You are in Standalone or have 1 player.	Play with 2+ players or Net Mode: Play As Client.
Client RepNotify / change event never fires	The value was set to what it already was.	Change the value, or tick Force Notify.
Spawned actor only appears in one window	It was spawned on a client.	Use MP Spawn Actor so it spawns on the server.
Send To Owner does nothing	The actor has no owning connection.	Set the actor's Owner to a player controller or pawn.
Late joiners never see an actor	Net Update Frequency is 0.	Set a positive Net Update Frequency.
Interactable state doesn't sync to others	Only Relevant To Owner is on.	Turn it off (the Interactable profile already sets relevancy).

Symptom	Likely cause	Fix
Set on a client has no effect	Client writes are disabled.	Server-authoritative by design; enable client writes in Settings if intended.
An effect only plays for the player who pressed the key	The Multicast is called on the client.	Call a Run on Server event, and the Multicast inside it.
The HUD only appears for the host	The widget is created in server code (Event Possessed).	Create it on the owning client, after Is Local Controller.
Get Game Mode is None	It is used on a client.	Use the GameState or PlayerState.
A variable is right on the server but 0 on clients	It is not replicated, or C++ is missing DOREPLIFETIME.	ScarNet names the variable; replicate it.
Players see two of something	A replicated actor is spawned inside a Multicast.	Spawn once on the server.

12 Testing multiplayer on one PC

- **Quick (PIE):** in the Play dropdown set **Number of Players = 2** and **Net Mode = Play As Client**. You get two windows with a hidden dedicated server underneath. ScarNet keeps each player's warnings separate even though they share one editor.
- **Realistic:** open **Tools** → **ScarNet** → **ScarNet Debugger** and use **Dedicated Server Testing**. One click launches a dedicated server (or listen host) plus N client processes of your current map, each with its own log in `Saved/Logs`.
- **Automatic reports:** add `-ScarNetReport=10` to any server or client command line and that process writes a full report to `Saved/ScarNet/` every 10 seconds (5 at the least). Useful for headless tests and build machines, and for comparing what the server and each client saw.

```
UnrealEditor.exe MyGame.uproject MyMap -server -log -ScarNetReport=10
UnrealEditor.exe MyGame.uproject 127.0.0.1 -game -log -ScarNetReport=10
```

TIP Put movement behind Switch Has Authority

When you test replication, only the **server** should drive the change (move the actor, set the variable). If both machines do it locally, both windows look right and you have proven nothing.

13 Settings worth knowing

Project Settings → Plugins → ScarNet:

- **Debugger:** hotkey (F9 by default), refresh interval, overlay interval, max rows.
- **Security:** allow or deny client spawn, destroy and variable-write requests. All validated on the server.
- **Profiles:** every value each built-in profile applies, editable per project.

- **Server Testing:** default client count, dedicated or listen, extra launch parameters.

Detection thresholds

Setting	Default	What it controls
Packet Loss Warn Percent	5	Packet loss above this shows a warning.
Always Relevant Warn Count	40	More Always Relevant actors than this shows a tip.
Reliable Rpc Warn Per Second	30	One Reliable RPC sent faster than this shows a warning.
Motion Desync Warn Cm	300	How far a client's copy may be from the server's before Out of sync.
Idle Actor Seconds	30	How long an awake actor must send nothing before the idle tip.

Quick FAQ

Question	Answer
Does it work with Steam / EOS?	Yes, with any Online Subsystem. ScarNet has no backend-specific code.
Does it run in packaged / shipping builds?	Yes. The debugger uses runtime Slate and F9 works in packaged games. The per-function RPC list needs a Development build. On a dedicated server use ScarNet.Report or -ScarNetReport.
Is it heavy?	No. With the debugger closed nothing runs. While it is open, a check every couple of seconds usually takes well under 1 ms, and RPC counting adds one small lookup per RPC.
How sure can I be of what it says?	Every check was tested in real dedicated-server and client processes against planted mistakes sitting next to correct setups that look almost the same. Runtime: 37 planted problems in C++ and Blueprint classes, on a clean network, a 60 ms network and a 120 ms network with 10% packet loss, 7 sessions: 314 of 315 findings were reported (the one miss was a value that packet loss had already put right, so there was nothing wrong to see) and none of the 18 correct setups was ever flagged in 175 reports. Project Check: 27 of 27 planted mistakes found, none of 21 correct look-alikes flagged. On a complete multiplayer game (129 Blueprints, 82 C++ RPCs, about 270 replicated actors) nothing that works correctly was reported as a problem. Movement and variable problems only count after a client confirms them twice.
Can I still write my own replication?	Absolutely. ScarNet sits on top of Unreal's system. Mix and match freely.

ScarNet - Multiplayer Automation & Debugger • Black Scar Studio • Unreal Engine 5.2 to 5.8.
 Support: discord.gg/a3QdMX9JZj