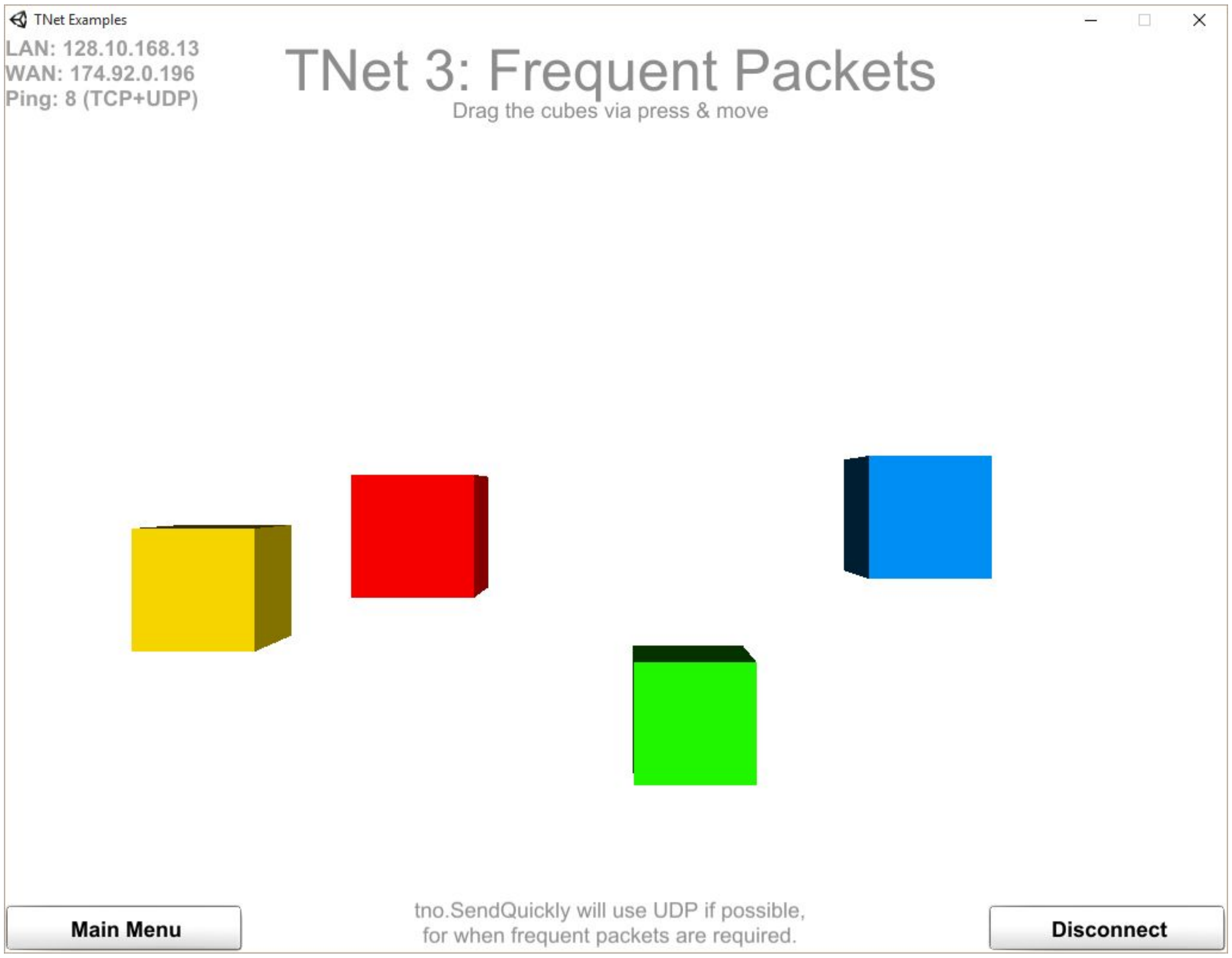
TNet 3: Frequent Packets In-Depth

The Frequent Packets example scene demonstrates how multiple people can interact with objects in a scene at the same time and have the drag information match on all clients.


TNet Examples
LAN: 128.10.168.13
WAN: 174.92.0.196
Ping: 8 (TCP+UDP)
TNet 3: Frequent Packets
Drag the cubes via press & move
tno.SendQuickly will use UDP if possible,
for when frequent packets are required.
Main Menu
Disconnect

Quick Overview

The Main Camera starts off with the `TouchHandler` script and broadcasts out messages when reacting to mouse clicks. The Dragged Cube objects are the ones which receive these broadcasts.

The Dragged Cubes have a couple scripts on them. The standard `TNObject` script which is used on all networked objects and the custom `DraggedObject` script which handles the dragging and dropping part.

Also included in the project is an object called “Improve Latency on Mobiles” which contains the `ImproveLatency` script. TNet is designed to work as fast as possible on PC’s, however one of the algorithms it uses can cause lag when used on mobile devices. This script disables that algorithm if it’s built for mobile.

Details

When a user clicks, the DraggedObject script on the cubes reacts to the OnPress event.

```
if (mOwner == null)
{
	// Call the claim function directly in order to make it feel more
responsive
	ClaimObject(TNManager.playerID, mTrans.position);

	// Inform everyone else
	tno.Send(2, Target.OthersSaved, TNManager.playerID, mTrans.position);
}
```

If the object doesn’t have an owner, then the client will notify everyone else that they are taking control of the object.

```
else if (mOwner == TNManager.player)
{
	// When the mouse or touch gets released, inform everyone that the player
no longer has control.
	ClaimObject(0, mTrans.position);
	tno.Send(2, Target.OthersSaved, 0, mTrans.position);
}
```

If the mouse button has been released and this client is the owner of this cube, then notify everyone that this client is relinquishing control.

```
[RFC(2)]
void ClaimObject (int playerID, Vector3 pos)
{
	mOwner = TNManager.GetPlayer(playerID);
	mTrans.position = pos;
	mTarget = pos;

	// Move the object to the Ignore Raycast layer while it's being dragged
	gameObject.layer = LayerMask.NameToLayer((mOwner != null) ? "Ignore
Raycast" : "Default");
}
```

`RFC(2) ClaimObject` finds the player which matches the `playerID` and records them as the owner of this object. If the `playerID` is 0, the owner will be null. Depending on the value in `mOwner`, the cube will be moved to/from the Ignore Raycast or Default layers. While dragging the cube, it needs to be on the Ignore Raycast layer or else the raycast going from the mouse cursor into the world would always hit the closest surface of the cube and it would appear to fly into the camera. By raycasting through the cube, the point will always be on the floor (Or the surface of an unowned cube).

```
void OnDrag (Vector2 delta)
{
        if (mOwner == TNManager.player)
        {
                mTarget = TouchHandler.worldPos;

                // Here we send the function via "SendQuickly", which is faster
        than regular "Send"
                // as it goes via UDP instead of TCP whenever possible. The
        downside of this approach
                // is that there is up to a 4% chance that the packet will get
        lost. However since
                // this update is sent so frequently, we simply don't care.
                tno.SendQuickly(3, Target.OthersSaved, mTarget);
        }
}
```

In the `OnDrag` event, the raycast from the mouse cursor to the floor is sent over the network to all other clients connected to the channel.

```
[RFC(3)] void MoveObject (Vector3 pos) { mTarget = pos; }
```

`RFC(3)` receives the updated position and records the value as the target point. Over the next few frames frame the current position of the cube will be moved towards `mTarget` in the `Update()` function.