Sprinting Logic

Due to flaws in the Minecraft protocol there is a sprinting desync. Resulting in inconsistent results despite timing sprint resets correctly. MMC has something that I like to call "W Tap Assist" that makes it way less time sensitive. This has the side effect of making it way less skill based, and way more cps dependent though, so it isn't a great solution.
Intel Edits makes the observation that you can have your W key released when you hit someone and still get a sprint hit. Which is not vanilla behaviour.
Below is the code that shows how extraKnockback is set.

```java
public void a(PacketPlayInEntityAction packetplayinentityaction) {
    if (ensureMainThread(packetplayinentityaction))
        return; // Blossom
    // CraftBukkit start
    if (this.player.dead)
        return;
    switch (packetplayinentityaction.b()) {
        case START_SNEAKING:
        case STOP_SNEAKING:
            PlayerToggleSneakEvent event = new PlayerToggleSneakEvent(this.getPlayer(),
                    packetplayinentityaction.b() == PacketPlayInEntityAction.EnumPlayerAction.
            this.server.getPluginManager().callEvent(event);

            if (event.isCancelled()) {
                return;
            }
            break;
        case START_SPRINTING:
        case STOP_SPRINTING:
            PlayerToggleSprintEvent e2 = new PlayerToggleSprintEvent(this.getPlayer(),
                    packetplayinentityaction.b() == PacketPlayInEntityAction.EnumPlayerAction.
            this.server.getPluginManager().callEvent(e2);

            if (e2.isCancelled()) {
                return;
            }
            break;
    }
    // CraftBukkit end
    this.player.resetIdleTimer();
    switch (PlayerConnection.SyntheticClass_1.b[packetplayinentityaction.b().ordinal()]) {
        case 1:
            this.player.setSneaking(true);
            break;

        case 2:
            this.player.setSneaking(false);
            break;
        case 3:
            this.player.extraKnockback = true;
            this.player.setSprinting(true);
            break;

        case 4:
            int ticksUntilOffHitCooldown = this.player.noDamageTicks - (this.player.maxNoDamag
            if (ticksUntilOffHitCooldown > 0)
                this.player.extraKnockback = false;
            this.player.setSprinting(false);
            break;

        case 5:
            this.player.a(false, true, true);
            // this.checkMovement = false; // CraftBukkit - this is handled in teleport
            break;

        case 6:
            if (this.player.vehicle instanceof EntityHorse) {
                ((EntityHorse) this.player.vehicle).v(packetplayinentityaction.c());
            }
            break;

        case 7:
            if (this.player.vehicle instanceof EntityHorse) {
                ((EntityHorse) this.player.vehicle).g((EntityHuman) this.player);
            }
            break;

        default:
            throw new IllegalArgumentException("Invalid client command!");
    }

}
```

Attack Buffer

If you are off with timing your hits, MMC allows 1 tick of buffer time where it will queue an attack for when you are off damage cooldown. You can verify this by using a macro that intentionally hits, and then times the next hit slightly before the no damage ticks run out.

Direction Logic

if you stand still and hit your opponent who is also standing still, and change where you are aiming, the direction you hit them will change. Normally the first stage of knockback is only based on relative direction, whereas the second stage (extra knockback) is fully yaw based. However there is a yaw based component to both stages. From testing I have concluded that each stage has 50/50 yaw and direction based components to it. Which would make sense seeing as it is an improvement over default logic.

Values

Analyzing velocity packets using pakkit, you can see which of these values correspond to what functionality. One of the findings was that the horizontal and vertical in this screenshot are actually switched round. Because the resulting vertical knockback from the packet was ~0.3622, and doing 0.9055 * 0.4 lined up with that perfectly. Below you can see the full logic written as a groovy script.

```
import net.minecraft.server.v1_8_R3.EntityLiving
import gg.mineral.server.combat.KnockbackProtocol

totalVertical = 0.9055d
totalHorizontal = 0.8835d
rangeFactor = 0.035d
maxReduction = 0.4d
startRange = 3.0d
idleReduction = 0.6d

attackBuffer = 1

horizontal = totalHorizontal * idleReduction
horizontalExtra = totalHorizontal * (1 - idleReduction)
vertical = totalVertical * 0.4d
verticalExtra = 0.0d

friction = 0.0d
verticalLimit = 0.4d
attackerSlowdown = 0.6d

protocol = new KnockbackProtocol() {

// Helper method to calculate range reduction
double calculateRangeReduction(double distance) {
    if (distance <= startRange) {
        return 0.0d
    }
    double modifiedRange = rangeFactor * (distance - maxReduction)
    return Math.min(modifiedRange, maxReduction)
}

void firstStage(EntityLiving attacker, EntityLiving victim) {
    // Apply friction
    if (friction != 0) {
        victim.motX /= friction
        victim.motY /= friction
        victim.motZ /= friction
    } else {
        victim.motX = 0
        victim.motY = 0
        victim.motZ = 0
    }

    // Calculate distance and range reduction
    double distanceX = attacker.locX - victim.locX
    double distanceZ = attacker.locZ - victim.locZ
    double distance = Math.sqrt(distanceX * distanceX + distanceZ * distanceZ)
    double rangeReduction = calculateRangeReduction(distance)

    // Apply modified horizontal knockback with range reduction
    double modifiedHorizontal = horizontal - rangeReduction

    double magnitude = Math.sqrt(distanceX * distanceX + distanceZ * distanceZ)

    // 50% displacement-based knockback
    victim.motX -= (distanceX / magnitude) * (modifiedHorizontal * 0.5d)
    victim.motZ -= (distanceZ / magnitude) * (modifiedHorizontal * 0.5d)

    // 50% yaw-based knockback
    double yaw = Math.toRadians(attacker.yaw)
    double sin = -Math.sin(yaw)
    double cos = Math.cos(yaw)
    victim.motX += sin * (modifiedHorizontal * 0.5d)
    victim.motZ += cos * (modifiedHorizontal * 0.5d)

    // Set vertical motion
    victim.motY = vertical

    if (victim.motY > verticalLimit) {
        victim.motY = verticalLimit
    }
}

void secondStage(EntityLiving attacker, EntityLiving victim, int knockbackEnchantLevel) {
    int extraKBMult = knockbackEnchantLevel

    // View MineralSpigot src to view how this is handled differently to isSprinting
    if (attacker.extraKnockback) {
        extraKBMult++
    }

    if (extraKBMult > 0) {
        double distanceX = attacker.locX - victim.locX
        double distanceZ = attacker.locZ - victim.locZ
        double distance = Math.sqrt(distanceX * distanceX + distanceZ * distanceZ)

        double modifiedExtraHorizontal = horizontalExtra * extraKBMult

        double magnitude = Math.sqrt(distanceX * distanceX + distanceZ * distanceZ)

        // 50% displacement-based knockback
        victim.motX -= (distanceX / magnitude) * (modifiedExtraHorizontal * 0.5d)
        victim.motZ -= (distanceZ / magnitude) * (modifiedExtraHorizontal * 0.5d)

        // 50% yaw-based knockback
        double yaw = Math.toRadians(attacker.yaw)
        double sin = -Math.sin(yaw)
        double cos = Math.cos(yaw)
        victim.motX += sin * (modifiedExtraHorizontal * 0.5d)
        victim.motZ += cos * (modifiedExtraHorizontal * 0.5d)

        victim.motY += verticalExtra

        attacker.motX *= attackerSlowdown
        attacker.motZ *= attackerSlowdown
        attacker.extraKnockback = false
    }
}
};
```