Manual – LAN Tool + iPerf3 WebUI

2026-02-14

Handbuch

Projekt: LAN Tool

Windows: `ip_setup.ps1` (Start: `Start_win.bat`)
Linux: `Start_Linux.sh`
+ iPerf3 Web-Oberfläche (Flask)

Ziel: Zwei PCs (PC A / PC B) für direkte LAN-Messungen vorbereiten und iPerf3-Tests komfortabel im Browser starten + auswerten.
Stand: 2026-02-14 (angepasst auf Windows **und** Linux)
Autor: (Fork) Jumbo125 — original by MaddyDev-glitch

Inhaltsverzeichnis

---

0) Was hat sich geändert?

Neu: Ausführbare Windows und Linux Dateien inkl. IP-Einstellung

- **IP-Setup** gibt es jetzt in zwei Varianten:
  - **Windows:** PowerShell-Menü `Setup_IP\ip_setup.ps1` (Start über `Start_win.bat`)
  - **Linux:** Bash-Menü `Start_Linux.sh` (setzt IP/Firewall/Start in neuem Terminal)

WebUI (Python/Flask) ist jetzt OS-agnostisch

- Läuft auf **Windows oder Linux**.
- iPerf3-Binary wird je nach OS/Architektur automatisch gewählt:
  - **Windows:** `..\IPERF\iperf3.exe` (falls vorhanden), sonst `iperf3` aus `PATH`
  - **Linux:** `..\IPERF\iperf3-amd64` oder `..\IPERF\iperf3-arm64v8` (macht `chmod +x`), sonst `iperf3` aus `PATH`

Stabilitäts-Änderungen (PATCH 2026-02-13)

- `/run_iperf` blockiert nicht mehr durch langsame Counter-Reads:
  - **Baseline-Counter werden im Worker-Thread** gelesen (nicht im HTTP-Handler).
- `run_cmd()` hat **Timeouts**, damit PowerShell/`ethtool` nicht ewig hängen.

Pro Run Logfile

- Jeder Test schreibt nach `logs/iperf_YYYYMMDD_HHMMSS.log`
- WebUI streamt zusätzlich Metainfo:
  - `CMD: ...`
  - `LOGFILE: ...`
  - `WORKER: ...`

IPERF Ausgabe mit `--json` Flag

CRC und FCS

- Physisches Kabelgebrechen

---

1) Rollen: PC A / PC B

Rolle	Gerät	Aufgabe
PC A	WebUI-PC	WebUI läuft (Flask GUI) und wird im Browser angezeigt
PC B	Server-PC	iPerf3 Server läuft im Terminal/Konsole (offen lassen)

Hinweis: Windows Linux kann beliebig gemischt werden.
Wichtig: **PC B muss der Server sein**, PC A startet den Client über die WebUI.

---

2) Voraussetzungen

Hardware

- 1× LAN-Kabel (direkt PC PC oder über Switch)
- Beide PCs im selben Netz (im Tool: `192.168.10.0/24`)

Software

- iPerf3 ist im Projekt enthalten oder im `PATH` verfügbar

- WebUI: Python-App (Flask) wird über Start-Skripte gestartet (oder manuell)

Rechte

- **Windows:** `Start_win.bat` verlangt Admin (prüft via `net session`)
- **Linux:** `Start_Linux.sh` verlangt `sudo`/root (`id -u == 0`)

---

3) Schnellstart (Generisch – gilt für alle OS-Kombis)

PC A (WebUI-PC)

1. **Adapter wählen**
2. **Backup erstellen**
3. **Firewall-Regeln setzen** (oder temporär deaktivieren – falls nötig)
4. **PC A IP setzen:** `192.168.10.1/24`
5. **PC A starten:** startet die WebUI (Flask)
6. Browser öffnen:
   - `http://192.168.10.1:5000`
   - oder `http://localhost:5000`

PC B (Server-PC)

1. **Adapter wählen**
2. **Backup erstellen**
3. **Firewall-Regeln setzen** (oder temporär deaktivieren)
4. **PC B IP setzen:** `192.168.10.2/24`
5. **PC B starten:** iPerf3 Server läuft im Terminal → **nicht schließen**

Nach dem Test (Cleanup)

- Firewall wieder aktivieren (falls deaktiviert)
- Firewall-Regeln entfernen
- Restore (Backup zurück) **oder** DHCP aktivieren

---

4) Windows Ablauf (Start über `Start_win.bat`)

Startscript: `Start_win.bat`

- Macht **Admin-Check** (ohne UAC prompt):
  - Wenn nicht Admin → Fehlermeldung + Hinweis „Als Administrator ausführen“
- Startet dann:
  - `powershell -NoProfile -ExecutionPolicy Bypass -File "%~dp0Setup_IP\ip_setup.ps1"`

Windows Quick Guide – PC A

1. `Start_win.bat` **als Administrator** starten
2. Im PowerShell-Menü:
   - `8) Adapter wechseln` (falls nötig)
   - `1) Backup schreiben`
   - `6) Firewall-Regel setzen`
   - optional `12) Windows Firewall DEAKTIVIEREN`
   - `4) PC A setzen` → `192.168.10.1/24`
   - `9) PC A starten` → öffnet neues CMD (WebUI/Starter)
3. Browser öffnen: URL aus dem CMD (typisch `http://192.168.10.1:5000`)

Windows Quick Guide – PC B

1. `Start_win.bat` **als Administrator** starten
2. Im Menü:
   - `8) Adapter wechseln`
   - `1) Backup schreiben`
   - `6) Firewall-Regel setzen`

- optional `12) Windows Firewall DEAKTIVIEREN`
- `5) PC B setzen` → `192.168.10.2/24`
- `10) PC B starten` → **CMD offen lassen** (iPerf3 Server)

3. Wenn fertig:
   - `11) Firewall AKTIVIEREN`
   - `7) Firewall-Regel löschen`
   - `2) Wiederherstellen` **oder** `3) DHCP aktivieren`

---

5) Linux Ablauf (Start über `Start_Linux.sh`)

Startscript: `Start_Linux.sh`

Das Script übernimmt: - Adapter-Auswahl (Pflicht beim Menüstart) - Backup/Restore als JSON (über „portable python“ oder system `python3`) - DHCP / Static IP (via `nmcli` wenn möglich, sonst Fallback `ip`/`dhclient`) - UFW Regeln: - TCP/UDP Port **5201** nur von `192.168.10.1` und `192.168.10.2` - ICMP Ping: schreibt einen Block in `/etc/ufw/before.rules` - Block-Marker: `# LAN_TOOL_BEGIN` bis `# LAN_TOOL_END` - Erstellt einmalig Backup: `/etc/ufw/before.rules.lan_tool.bak` - Start PC A/B in neuem Terminal (wenn möglich): - `Setup_IP/Start_PC_A_Linux.sh` - `Setup_IP/Start_PC_B_Linux.sh` - **Diese Start-Skripte werden NICHT mehr generiert** – sie müssen fix existieren.

> **Hinweis:** Bei SSH warnt das Script, weil IP-Änderungen die Verbindung trennen können.

Linux Quick Guide – PC A

1. Start:

```
sudo ./Start_Linux.sh
```

2. Adapter auswählen
3. Menü:
   - `1) Backup schreiben`
   - `6) UFW Regeln setzen` (oder Firewall manuell)
   - `4) PC A setzen (192.168.10.1/24)`
   - `9) PC A starten` → neues Terminal → WebUI
4. Browser: `http://192.168.10.1:5000`

Linux Quick Guide – PC B

1. Start:

```
sudo ./Start_Linux.sh
```

2. Adapter auswählen
3. Menü:
   - `1) Backup schreiben`
   - `6) UFW Regeln setzen`
   - `5) PC B setzen (192.168.10.2/24)`
   - `10) PC B starten` → neues Terminal → iPerf3 Server **laufen lassen**
4. Cleanup:
   - `7) UFW Regeln löschen`
   - `2) Wiederherstellen` **oder** `3) DHCP`

---

6) Was startet „PC A starten“ / „PC B starten“?

Windows

Wie bisher: euer PowerShell-Tool öffnet per BAT ein **neues `cmd.exe`** (typisch via `Start-Process cmd.exe /k ...`): - **PC A:** startet WebUI/Flask-Starter - **PC B:** startet iPerf3 Server

Linux

`Start_Linux.sh` startet fix vorhandene Scripts – möglichst in neuem Terminal: - `Setup_IP/Start_PC_A_Linux.sh` → WebUI/Flask - `Setup_IP/Start_PC_B_Linux.sh` → iPerf3 Server

> Falls kein Terminal-Emulator gefunden wird (gnome-terminal/konsole/xfce4-terminal/…/xterm), läuft es im aktuellen Terminal.

---

7) WebUI Bedienung (PC A)

7.1 Browser öffnen

Typisch: - `http://localhost:5000` (lokal am PC A) - `http://192.168.10.1:5000` (LAN-IP)

> Standard-Bind ist `0.0.0.0:5000` (konfigurierbar in `settings.json`).

7.2 Interface auswählen

Dropdown „Interface“ nutzt `/api/interfaces`:

- **Windows:** PowerShell `Get-NetAdapter` (Timeout 15s) + Fallback via CIM (`Win32_NetworkAdapter`)
- **Linux:** `ip -o link show` (ohne `lo`)

7.3 Target & Port

- Target für PC A → PC B: `192.168.10.2`
- Port: `5201`

7.4 Mode

- **Upload:** PC A sendet → PC B empfängt
- **Download:** Reverse Mode (`-R`) → PC B sendet Richtung PC A

7.5 Run (Speed Test)

Beim Start ruft die UI `/run_iperf` auf und öffnet danach/parallel den Stream `/stream_iperf`.

Wichtig aus dem Code: - Bei jedem Run wird der alte Prozess beendet (`terminate()` → `kill()` falls nötig) - Queue wird zurückgesetzt → Stream liefert Run-Daten dieses einen aktiven Runs - Stream sendet regelmäßig Keepalive `ping`

---

8) Wie der iPerf-Run technisch gebaut wird (aus Python-Code)

iPerf3 Command

Basis: - `iperf3 -c <target> -p <port> -P <streams> -i <interval> -t <duration>`

Zusätze: - `--json-stream` (damit die WebUI Interval-Werte sauber streamen kann) - `--forceflush` (Ausgabe wird sofort geflusht) - `--connect-timeout <ms>`
- Default **3000ms** → verhindert „silent hang“ wenn Server nicht erreichbar

UDP: - `-u -b <bandwidth>`

Download: - `-R`

Units / Anzeige

Die WebUI rechnet `bits_per_second` aus dem JSON-Stream auf: - `Kbits`, `Mbits`, `Gbits`

Logfiles

Pro Run: - `logs/iperf_YYYYMMDD_HHMMSS.log`

Zusätzlich streamt die WebUI: - `LOGFILE: <pfad>`

---

9) Stats / Fehler / Link-Info (Windows vs. Linux)

9.1 Link-Info

- **Windows:** `Get-NetAdapter -Name "<iface>" | Select Name, Status, LinkSpeed`
- **Linux:** `ethtool <iface>` (Speed / Duplex / Link detected / Auto-negotiation)

9.2 Counter (für Δ-Berechnung)

> **Wichtig:** Baseline wird im Worker beim Run-Start gespeichert, `/api/stats` liefert danach Delta = `now - baseline`.

Windows (zuverlässig, aber aggregiert): - `ReceivedErrors` - `OutboundErrors` - `ReceivedDiscarded` - `OutboundDiscarded`

Linux (abhängig vom Treiber, oft detaillierter): - `rx_crc_errors`, `rx_fcs_errors`, `rx_errors`, `tx_errors` - `rx_dropped`, `tx_dropped`, `rx_missed_errors`, … - gelesen über `ethtool -S <iface>`

> CRC/FCS ist unter Linux **eher** möglich (treiberabhängig), unter Windows meist **nicht separat**.

---

10) Konfiguration: `settings.json` und `env.yaml`

`settings.json` (neben der Python-App)

Falls nicht vorhanden, gelten Defaults: - `web_host: 0.0.0.0` - `web_port: 5000` - `iperf_port: 5201` - `default_target: ""` - `default_iface: ""`

Beispiel:

```
{
  "web_host": "0.0.0.0",
  "web_port": 5000,
  "iperf_port": 5201,
  "default_target": "192.168.10.2",
  "default_iface": "Ethernet 2"
}
```

`env.yaml`

Wird genutzt für UI-Branding: - `logos: [...]` - `theme: {...}`

---

11) Checkliste vor dem Test (Windows & Linux)

- ☐ Richtiger Adapter gewählt (auf beiden PCs)
- ☐ Backup erstellt (`lan_backup_<iface>.json`)
- ☐ Firewall-Regeln gesetzt (oder temporär deaktiviert)
- ☐ PC A IP: `192.168.10.1/24`
- ☐ PC B IP: `192.168.10.2/24`
- ☐ PC B Server läuft (Terminal/Konsole offen)
- ☐ WebUI erreichbar (PC A: `http://192.168.10.1:5000`)
- ☐ Target in WebUI: `192.168.10.2`, Port `5201`
- ☐ Interface in WebUI korrekt gesetzt (für Counter/Delta)

---

12) Nach dem Test (Cleanup)

Windows

- ☐ `11) Firewall aktivieren` (falls deaktiviert)
- ☐ `7) Firewall-Regel löschen`
- ☐ `2) Wiederherstellen` **oder** `3) DHCP aktivieren`

Linux

- ☐ `7) UFW Regeln löschen` (entfernt auch den ICMP-Block im `before.rules`)
- ☐ `2) Wiederherstellen` **oder** `3) DHCP aktivieren`

---

13) Typische Fehler & Lösungen (aktualisiert)

Interfaces leer in der WebUI

- **Windows:** PowerShell kann kalt sehr langsam sein → Geduld / Reload
- **Windows:** Fallback über CIM ist drin, kann aber leere Liste liefern, wenn Adapter-Infos fehlen
- **Linux:** `ip link` zeigt nur echte Interfaces, `lo` wird gefiltert → Interface muss existieren / up sein

Client hängt / kein Output beim Start

- iPerf3 JSON-Stream kann bis zum Connect **stumm** bleiben → daher `--connect-timeout` (Default 3000ms)
- Prüfen:
  - PC B Server wirklich gestartet?
  - Firewall blockt Port 5201?
  - Target-IP stimmt (`192.168.10.2`)?

Ping geht, iPerf nicht

- Port 5201 TCP/UDP freigeben (Windows-Regel oder UFW)
- Linux: UFW aktiv? → `ufw status`
- Notfall: Firewall kurz deaktivieren (nur zum Gegencheck)

SSH Verbindung weg nach IP setzen (Linux)

- Script warnt: bei SSH IP ändern → Verbindung kann abbrechen
  → am besten lokal am Gerät oder mit Out-of-band Zugriff arbeiten.

Fehlerindikatoren: Kabelbruch, zu langes Kabel, schlechte Stecker/Knicks (CRC/FCS & Co.)

Diese WebUI ist **kein Kabel-Zertifizierer**, aber sie kann **unter Last** typische Symptome sichtbar machen, die stark auf ein physikalisches Problem hindeuten (Kabelbruch/angescheuerte Ader, zu enge Biegung, zu lang/zu schlechte Qualität, wackeliger RJ45, schlechte Patchpanel-Buchse, Störungen/Nahnebensprechen).

„Harmlos“ vs. „Verdächtig“

Harmlos / normal: - Delta-Counter bleiben bei 0 (oder extrem seltene Einzelwerte, die nicht weiter steigen) - iPerf-Durchsatz ist stabil (kaum Schwankung) und nahe der erwarteten Linkrate

Verdächtig: - **Delta-Counter steigen kontinuierlich** während eines iPerf Runs - Durchsatz schwankt stark, obwohl CPU/Settings konstant sind - Link-Speed fällt z.B. von 1 Gbit auf 100 Mbit (Auto-Negotiation / schlechtes Pair)

Welche Counter sind aussagekräftig?

Linux (`ethtool -S`) – beste Chance auf echte CRC/FCS Hinweise Typische Keys (treiberabhängig): - `rx_crc_errors` / `rx_fcs_errors` **starker Hinweis auf Signal-/Kabelproblem** - `rx_errors` / `tx_errors` „irgendwas geht kaputt“ (PHY, Treiber, Kabel, Duplex/Speed-Mismatch) - `rx_dropped` / `tx_dropped` kann auch durch Queue/CPU entstehen (nicht nur Kabel) - `rx_missed_errors` oft „Receiver overrun“ (Last/Interrupts), nicht zwingend Kabel

Daumenregel:
Wenn **CRC/FCS (`rx_crc_errors/rx_fcs_errors`)** während Last ansteigt, ist die Wahrscheinlichkeit hoch, dass es **physikalisch** ist (Kabel/Stecker/Störung).

Windows (`Get-NetAdapterStatistics`) – aggregiert, aber trotzdem nützlich

- `ReceivedErrors` / `OutboundErrors` echte Fehler (aber ohne CRC-Detail)
- `ReceivedDiscarded` / `OutboundDiscarded` kann auch durch Treiber/Queues/Last entstehen

Daumenregel:
Wenn Errors **nur unter Last** steigen (bei iPerf), ist das oft ein physikalischer Hinweis – besonders wenn es mit Durchsatz-Einbrüchen zusammenfällt.

iPerf3 Symptome, die zu „Kabel/Signal“ passen

TCP (Standard)

- Durchsatz bricht ein, „sägezahnartig“
- Viele Retransmits (in iPerf-Endsummary sichtbar; in euren Logs ebenfalls, weil das komplette JSON-Stream-Output geloggt wird)
- Verbindungen brechen ab / Timeout / „unable to connect“ (bei schweren Problemen oder Link-Flaps)

Hinweis: Eure WebUI streamt primär Bandwidth-Werte.
Für Detailanalyse: öffne das Run-Logfile (`logs/iperf_*.log`) und suche im `end`-Event nach Feldern wie `retransmits`.

UDP

- Packet Loss steigt (bei ausreichend hoher `-b` Rate)
- Jitter ist auffällig hoch/instabil

„Kabel zu lang“ – was ist realistisch?

- Ethernet über Kupfer ist normativ je nach Kategorie typischerweise bis ~100m (Permanent Link + Patch).
- In der Praxis führen *billige, dünne, schlecht geschirmte, beschädigte oder stark geknickte* Kabel deutlich früher zu Problemen.
- „Zu lang“ äußert sich oft nicht als kompletter Ausfall, sondern als **CRC/FCS-Fehler unter Last** und **Link-Speed-Downgrades**.

Quick-Diagnose (5 Minuten)

1. **iPerf Run starten** (10–30s, ggf. 4–8 Streams)
2. Währenddessen `/api/stats` beobachten (WebUI):
   - Steigen **Errors/CRC/FCS**?
3. Wenn ja:
   - Kabel tauschen (kurz + qualitativ gut)
   - Anderen Port / andere Buchse testen
   - Stecker prüfen (wackelt? Rastnase? Knickschutz?)
   - Kabel nicht „auf Zug“ und keine engen 90° Knicke
4. Wenn mit Ersatzkabel alles stabil → sehr wahrscheinlich physikalisch.

   Für eine wirklich eindeutige Aussage (NEXT/Return Loss/Länge/Pair Map) brauchst du einen Kabel-Zertifizierer oder TDR-Tester. Die WebUI ist ein pragmatischer Lasttest + Fehlerindikator.

Dokument-Version

2026-02-14

Manual

Project: LAN Tool

Windows: `ip_setup.ps1` (Start: `Start_win.bat`)
Linux: `Start_Linux.sh`
+ iPerf3 Web UI (Flask)

Goal: Prepare two PCs (PC A / PC B) for direct LAN measurements and run/evaluate iPerf3 tests conveniently in the browser.
Status: 2026-02-14 (adapted for Windows **and** Linux)
Author: (Fork) Jumbo125 — original by MaddyDev-glitch

Table of contents

---

0) What changed?

New: runnable Windows and Linux files incl. IP configuration

- **IP setup** now exists in two variants:
  - **Windows:** PowerShell menu `Setup_IP\ip_setup.ps1` (started via `Start_win.bat`)
  - **Linux:** Bash menu `Start_Linux.sh` (sets IP/firewall/start in a new terminal)

WebUI (Python/Flask) is now OS-agnostic

- Runs on **Windows or Linux**.
- iPerf3 binary is selected automatically based on OS/architecture:
  - **Windows:** `..\IPERF\iperf3.exe` (if present), otherwise `iperf3` from `PATH`
  - **Linux:** `..\IPERF\iperf3-amd64` or `..\IPERF\iperf3-arm64v8` (does `chmod +x`), otherwise `iperf3` from `PATH`

Stability changes (PATCH 2026-02-13)

- `/run_iperf` no longer blocks due to slow counter reads:
  - **Baseline counters are read in the worker thread** (not in the HTTP handler).

- `run_cmd()` has **timeouts** so PowerShell/`ethtool` can't hang forever.

Per-run logfile

- Each test writes to `logs/iperf_YYYYMMDD_HHMMSS.log`
- WebUI additionally streams meta info:
  - `CMD: ...`
  - `LOGFILE: ...`
  - `WORKER: ...`

iPerf output with `--json` flag

CRC and FCS

- Physical cable damage

---

1) Roles: PC A / PC B

Role	Device	Responsibility
PC A	WebUI PC	WebUI runs (Flask GUI) and is shown in the browser
PC B	Server PC	iPerf3 server runs in a terminal/console (keep it open)

Note: You can mix Windows Linux freely.
The only requirement is: **PC B must be the server**, PC A starts the client via the WebUI.

---

2) Prerequisites

Hardware

- 1× LAN cable (direct PC PC or via switch)
- Both PCs in the same subnet (tool uses `192.168.10.0/24`)

Software

- iPerf3 is included in the project or available via `PATH`
- WebUI: Python app (Flask) started via your scripts (or manually)

Privileges

- **Windows:** `Start_win.bat` requires admin (checks via `net session`)
- **Linux:** `Start_Linux.sh` requires `sudo`/root (`id -u == 0`)

---

3) Quick start (OS-agnostic – applies to all OS combinations)

PC A (WebUI PC)

1. **Select adapter**
2. **Create backup**
3. **Set firewall rules** (or temporarily disable, if needed)
4. **Set PC A IP:** `192.168.10.1/24`
5. **Start PC A:** launches the WebUI (Flask)
6. Open browser:
   - `http://192.168.10.1:5000`
   - or `http://localhost:5000`

PC B (Server PC)

1. **Select adapter**
2. **Create backup**
3. **Set firewall rules** (or temporarily disable)
4. **Set PC B IP:** `192.168.10.2/24`
5. **Start PC B:** iPerf3 server runs in terminal → **do not close**

After the test (cleanup)

- Re-enable firewall (if disabled)
- Remove firewall rules
- Restore (from backup) **or** re-enable DHCP

---

4) Windows flow (Start via `Start_win.bat`)

Start script: `Start_win.bat`

- Performs an **admin check** (without triggering a UAC prompt):
  - If not admin → error + hint "Run as administrator"
- Then starts:
  - `powershell -NoProfile -ExecutionPolicy Bypass -File "%~dp0Setup_IP\ip_setup.ps1"`

Windows quick guide – PC A

1. Run `Start_win.bat` **as Administrator**
2. In the PowerShell menu:
   - `8) Change adapter` (if needed)
   - `1) Write backup`
   - `6) Add firewall rule`
   - optional `12) DISABLE Windows Firewall`
   - `4) Set PC A` → `192.168.10.1/24`
   - `9) Start PC A` → opens a new CMD (WebUI/starter)
3. Open browser: URL shown in CMD (typically `http://192.168.10.1:5000`)

Windows quick guide – PC B

1. Run `Start_win.bat` **as Administrator**
2. In the menu:
   - `8) Change adapter`
   - `1) Write backup`
   - `6) Add firewall rule`
   - optional `12) DISABLE Windows Firewall`
   - `5) Set PC B` → `192.168.10.2/24`
   - `10) Start PC B` → **keep CMD open** (iPerf3 server)
3. When finished:
   - `11) ENABLE Firewall`
   - `7) Remove firewall rule`
   - `2) Restore` **or** `3) Enable DHCP`

---

5) Linux flow (Start via `Start_Linux.sh`)

Start script: `Start_Linux.sh`

This script handles: - Adapter selection (required at menu start) - Backup/restore as JSON (via "portable python" or system `python3`) - DHCP / static IP (via `nmcli` if possible, otherwise fallback to `ip`/`dhclient`) - UFW rules: - TCP/UDP port **5201** only from `192.168.10.1` and `192.168.10.2` - ICMP ping: writes a block into `/etc/ufw/before.rules` - Block markers: `# LAN_TOOL_BEGIN` to `# LAN_TOOL_END` - One-time backup: `/etc/ufw/before.rules.lan_tool.bak` - Starts PC A/B in a new terminal (if possible):

- `Setup_IP/Start_PC_A_Linux.sh` - `Setup_IP/Start_PC_B_Linux.sh` - **These start scripts are NO LONGER generated** – they must already exist.

> **Note:** Over SSH, the script warns because changing IPs may drop the connection.

Linux quick guide – PC A

1. Start:
   ```
   sudo ./Start_Linux.sh
   ```
2. Select adapter
3. Menu:
   - `1) Write backup`
   - `6) Add UFW rules` (or manage firewall manually)
   - `4) Set PC A (192.168.10.1/24)`
   - `9) Start PC A` → new terminal → WebUI
4. Browser: `http://192.168.10.1:5000`

Linux quick guide – PC B

1. Start:
   ```
   sudo ./Start_Linux.sh
   ```
2. Select adapter
3. Menu:
   - `1) Write backup`
   - `6) Add UFW rules`
   - `5) Set PC B (192.168.10.2/24)`
   - `10) Start PC B` → new terminal → keep iPerf3 server running
4. Cleanup:
   - `7) Remove UFW rules`
   - `2) Restore` **or** `3) Enable DHCP`

---

6) What does "Start PC A" / "Start PC B" launch?

Windows

As before: your PowerShell tool opens a **new `cmd.exe`** (typically via `Start-Process cmd.exe /k ...`): - **PC A:** starts WebUI/Flask starter - **PC B:** starts iPerf3 server

Linux

`Start_Linux.sh` launches fixed existing scripts – preferably in a new terminal: - `Setup_IP/Start_PC_A_Linux.sh` → WebUI/Flask - `Setup_IP/Start_PC_B_Linux.sh` → iPerf3 server

> If no terminal emulator is found (gnome-terminal/konsole/xfce4-terminal/…/xterm), it runs in the current terminal.

---

7) WebUI usage (PC A)

7.1 Open the browser

Typical: - `http://localhost:5000` (local on PC A) - `http://192.168.10.1:5000` (LAN IP)

> Default bind is `0.0.0.0:5000` (configurable in `settings.json`).

7.2 Select interface

The “Interface” dropdown uses `/api/interfaces`: - **Windows:** PowerShell `Get-NetAdapter` (timeout 15s) + fallback via CIM (`Win32_NetworkAdapter`) - **Linux:** `ip -o link show` (excluding `lo`)

7.3 Target & port

- Target for PC A → PC B: `192.168.10.2`
- Port: `5201`

7.4 Mode

- **Upload:** PC A sends → PC B receives
- **Download:** Reverse mode (`-R`) → PC B sends toward PC A

7.5 Run (speed test)

On start, the UI calls `/run_iperf` and then/parallel opens the stream `/stream_iperf`.

Important from the code: - Each run terminates the old process (`terminate()` → `kill()` if needed) - Queue is reset → stream shows data for the one active run - Stream sends regular keepalive `ping`

---

8) How an iPerf run is built (from Python code)

iPerf3 command

Base: - `iperf3 -c <target> -p <port> -P <streams> -i <interval> -t <duration>`

Extras: - `--json-stream` (so the WebUI can cleanly stream interval values) - `--forceflush` (output is flushed immediately) - `--connect-timeout <ms>`
- Default **3000ms** → prevents “silent hang” if the server isn’t reachable

UDP: - `-u -b <bandwidth>`

Download: - `-R`

Units / display

The WebUI converts `bits_per_second` from the JSON stream into: - `Kbits`, `Mbits`, `Gbits`

Logfiles

Per run: - `logs/iperf_YYYYMMDD_HHMMSS.log`

The WebUI also streams: - `LOGFILE: <path>`

---

9) Stats / errors / link info (Windows vs. Linux)

9.1 Link info

- **Windows:** `Get-NetAdapter -Name "<iface>" | Select Name, Status, LinkSpeed`
- **Linux:** `ethtool <iface>` (Speed / Duplex / Link detected / Auto-negotiation)

9.2 Counters (for Δ calculation)

> **Important:** The baseline is stored in the worker at run start; `/api/stats` returns delta = `now - baseline`.

Windows (reliable, but aggregated): - `ReceivedErrors` - `OutboundErrors` - `ReceivedDiscarded` - `OutboundDiscarded`

Linux (driver-dependent, often more detailed): - `rx_crc_errors`, `rx_fcs_errors`, `rx_errors`, `tx_errors` - `rx_dropped`, `tx_dropped`, `rx_missed_errors`, … - read via `ethtool -S <iface>`

CRC/FCS is **more likely** to be available on Linux (driver-dependent); on Windows it's usually **not** separated.

---

10) Configuration: `settings.json` and `env.yaml`

`settings.json` (next to the Python app)

If not present, defaults apply: - `web_host`: `0.0.0.0` - `web_port`: `5000` - `iperf_port`: `5201` - `default_target`: `""` - `default_iface`: `""`

Example:

```
{
  "web_host": "0.0.0.0",
  "web_port": 5000,
  "iperf_port": 5201,
  "default_target": "192.168.10.2",
  "default_iface": "Ethernet 2"
}
```

`env.yaml`

Used for UI branding: - `logos: [...]` - `theme: {...}`

---

11) Pre-test checklist (Windows & Linux)

- ☐ Correct adapter selected (on both PCs)
- ☐ Backup created (`lan_backup_<iface>.json`)
- ☐ Firewall rules set (or temporarily disabled)
- ☐ PC A IP: `192.168.10.1/24`
- ☐ PC B IP: `192.168.10.2/24`
- ☐ PC B server running (terminal/console open)
- ☐ WebUI reachable (PC A: `http://192.168.10.1:5000`)
- ☐ Target in WebUI: `192.168.10.2`, port `5201`
- ☐ Correct interface selected in WebUI (for counter/delta)

---

12) After the test (cleanup)

Windows

- ☐ `11) Enable Firewall` (if it was disabled)
- ☐ `7) Remove firewall rule`
- ☐ `2) Restore` **or** `3) Enable DHCP`

Linux

- ☐ `7) Remove UFW rules` (also removes the ICMP block in `before.rules`)
- ☐ `2) Restore` **or** `3) Enable DHCP`

---

13) Common issues & fixes (updated)

Interfaces list is empty in the WebUI

- **Windows:** PowerShell can be slow on a cold start → wait / reload
- **Windows:** CIM fallback exists, but can still return empty if adapter info is missing
- **Linux:** `ip link` only shows real interfaces, `lo` is filtered → interface must exist / be up

Client hangs / no output at start

- iPerf3 JSON stream can be silent until connect → hence `--connect-timeout` (default 3000ms)
- Check:
  - Is PC B server really started?
  - Is the firewall blocking port 5201?
  - Is the target IP correct (`192.168.10.2`)?

Ping works, iPerf does not

- Allow TCP/UDP port 5201 (Windows firewall rule or UFW)
- Linux: is UFW active? → `ufw status`
- Emergency check: temporarily disable firewall (only to validate)

SSH connection lost after setting IP (Linux)

- Script warns: changing IP over SSH can drop the connection
  → best to work locally or use out-of-band access.

Indicators: cable break, too long cable, bad connectors/bends (CRC/FCS & etc.)

This WebUI is **not** a cable certifier, but under **load** it can expose symptoms that strongly suggest a physical issue (broken conductor, sharp bend, cable too long/low quality, loose RJ45, bad patch panel jack, interference/crosstalk).

"Harmless" vs. "Suspicious"

Harmless / normal: - Delta counters stay at 0 (or very rare single events that don't keep increasing) - iPerf throughput is stable (minimal fluctuation) and close to expected link rate

Suspicious: - **Delta counters increase continuously** during an iPerf run - Throughput fluctuates heavily even though CPU/settings are constant - Link speed drops (e.g., from 1 Gbit to 100 Mbit) due to autonegotiation / bad pair

Which counters are meaningful?

Linux (`ethtool -S`) – best chance for true CRC/FCS signals Typical keys (driver-dependent): - `rx_crc_errors` / `rx_fcs_errors` **strong indicator of signal/cable issue** - `rx_errors` / `tx_errors` "something is failing" (PHY, driver, cable, duplex/speed mismatch) - `rx_dropped` / `tx_dropped` can also be queue/CPU related (not only cable) - `rx_missed_errors` often receiver overrun (load/interrupts), not necessarily cable

Rule of thumb:
If **CRC/FCS (`rx_crc_errors/rx_fcs_errors`)** increases under load, it's highly likely to be **physical** (cable/connector/interference).

Windows (`Get-NetAdapterStatistics`) – aggregated, but still useful

- `ReceivedErrors` / `OutboundErrors` real errors (but no CRC detail)
- `ReceivedDiscarded` / `OutboundDiscarded` can also be driver/queue/load related

Rule of thumb:
If errors rise **only under load** (during iPerf), that often points to a physical issue—especially if it correlates with throughput drops.

iPerf3 symptoms that match "cable/signal" problems

TCP (default)

- Throughput drops in a "sawtooth" pattern
- Many retransmits (visible in iPerf end summary; also in your logs because the full JSON stream output is logged)
- Connections drop / timeout / "unable to connect" (for severe issues or link flaps)

Note: Your WebUI mainly streams bandwidth values.
For deeper analysis: open the run logfile (`logs/iperf_*.log`) and search the `end` event for fields like `retransmits`.

UDP

- Packet loss increases (at sufficiently high `-b` rate)
- Jitter is notably high/unstable

"Cable too long" – what's realistic?

- Copper Ethernet is typically specified up to ~100m (permanent link + patch), depending on the category.
- In practice, *cheap, thin, poorly shielded, damaged, or sharply bent* cables can fail much earlier.
- "Too long" often shows up not as a full outage, but as **CRC/FCS errors under load** and **link-speed downgrades**.

Quick diagnosis (5 minutes)

1. Start an **iPerf run** (10–30s, optionally 4–8 streams)
2. While it runs, watch `/api/stats` in the WebUI:
   - Are **Errors/CRC/FCS** increasing?
3. If yes:
   - Swap the cable (short + good quality)
   - Test another port / jack
   - Check the connectors (loose? broken latch? kinked strain relief?)
   - Avoid tension and tight 90° bends
4. If a spare cable is stable → very likely physical.

   For an unambiguous statement (NEXT/Return Loss/length/pair map) you need a cable certifier or TDR tester. The WebUI is a pragmatic load test + error indicator.

Document version

2026-02-14