// Zwischenbericht: SPS-Benchmarking

Team

- Allmer
- Bierbaum
- Herbst

1. Aufgabenstellung

Ziel ist die Entwicklung eines Performance-Tests für drei Kommunikationsprotokolle im Zusammenspiel mit einer Siemens-S7-SPS:

Protokoll	Bibliothek
OPC UA	`opcua`
Siemens Web API	`requests`
S7 (proprietär)	`python-snap7`

Benchmarks:

- Schreiben einzelner Variablen (Bool, Int16, Int32) mit 1, 10 bzw. 20 Ops/s
- Übertragen eines Datenblocks mit ca. 1 kB (100 × LTime-Werte)

2. Entwicklungsumgebung

Dependencies sind in `pyproject.toml` gelistet. Als Package und Project-Manager wurde UV verwendet.

Konfiguration

Alle Verbindungsparameter werden über ein `.env` -File konfiguriert (siehe `.env.example`)

3. Softwarearchitektur

Adapter-Pattern

Das Adapter-Pattern ermöglicht es, die Benchmark-Logik von den protokollspezifischen Details zu trennen. Dadurch können alle drei Protokolle mit identischen Testfällen verglichen werden, ohne den Testcode zu duplizieren.

Für jedes Protokoll wurde ein Adapter implementiert, der eine gemeinsame abstrakte Basisklasse implementiert:

```
benchmark/
├── adapters/
│   ├── base.py      # GenericAdapter (ABC)
│   ├── webapi.py    # SpsWebApiAdapter
│   ├── opcua.py     # OpcUaAdapter
│   └── s7.py        # S7Adapter
├── benchmark_runner.py
└── main.py
```

Besonderheiten der Implementierung

Adapter	Besonderheit
WebAPI	Token-basierte Authentifizierung, SSL (verify=False für Self-Signed)
OPC UA	Automatische Typ-Erkennung (Bool/Int16/Int32) anhand Variablenname
S7	Direkter Speicherzugriff via `db_read` / `db_write` , konfigurierbare Offsets

Single-Writes

Beim Single-Write-Test wird eine einzelne Variable wiederholt mit Werten beschrieben:

```
# Bool: True/False im Wechsel
value = operations % 2 == 0

# Int16/Int32: Inkrementierender Wert
value = operations % 1000
```

Bulk-Daten

Es wird ein Array mit 100 Einträgen im Siemens-LTime-Format erzeugt (je 8 Bytes = 800 Bytes):

```
bulk_data = [f"LT#{i * 1000000}ns" for i in range(100)]
# Ergebnis: ["LT#0ns", "LT#1000000ns", "LT#2000000ns", ...]
```

4. Datenübertragungssicherheit

Alle Protokolle basieren auf TCP. Bei einer erfolgreichen Response wird angenommen, dass die Daten korrekt übertragen wurden. Eine explizite Verifikation im SPS-Speicher erfolgt **nicht**.

5. Benutzung

Voraussetzungen

1. Python installiert
2. Siemens-SPS erreichbar im Netzwerk
3. Datenblock `PerformanceData` (DB7) auf der SPS konfiguriert
   (Hinweis: Auf der SPS falsch als `PerformaceData` geschrieben)

Installation

```
cd benchmark
uv sync
cp .env.example .env
# .env mit korrekten Werten anpassen
```

Benchmark ausführen

```
cd benchmark
uv run python main.py
```

Ausgabe

Nach Abschluss werden im Verzeichnis `results_YYYYMMDD_HHMMSS/` folgende Dateien erzeugt:

Datei	Inhalt
`protocol_comparison.txt`	Tabellarischer Vergleich aller Protokolle
`comparison_latency.png`	Latenz-Vergleich (P50/P90/P99)
`comparison_ops.png`	Durchsatz-Vergleich (Ops/s)
`webapi/`, `opc_ua/`, `s7/`	Protokoll-spezifische Ergebnisse

6. Gemessene Metriken

Metrik	Beschreibung
Ops/s	Operationen pro Sekunde (erreichte Rate)
Latency P50	Median der Antwortzeiten
Latency P90	90. Perzentil
Latency P99	99. Perzentil
Throughput (kB/s)	Datendurchsatz bei Bulk-Writes

7. Aktuelle Erkenntnisse

Testergebnisse von 2026-01-07

Test	WebAPI	OPC UA	S7	Sieger
Bool @ 20 ops/s	3.23 ops/s	20.17 ops/s	20.17 ops/s	S7/OPC UA
Int16 @ 20 ops/s	3.25 ops/s	20.16 ops/s	20.18 ops/s	S7/OPC UA
Int32 @ 20 ops/s	3.32 ops/s	20.16 ops/s	20.17 ops/s	S7/OPC UA
Bulk (100 LTime)	3.24 ops/s	5.31 ops/s	101.62 ops/s	S7

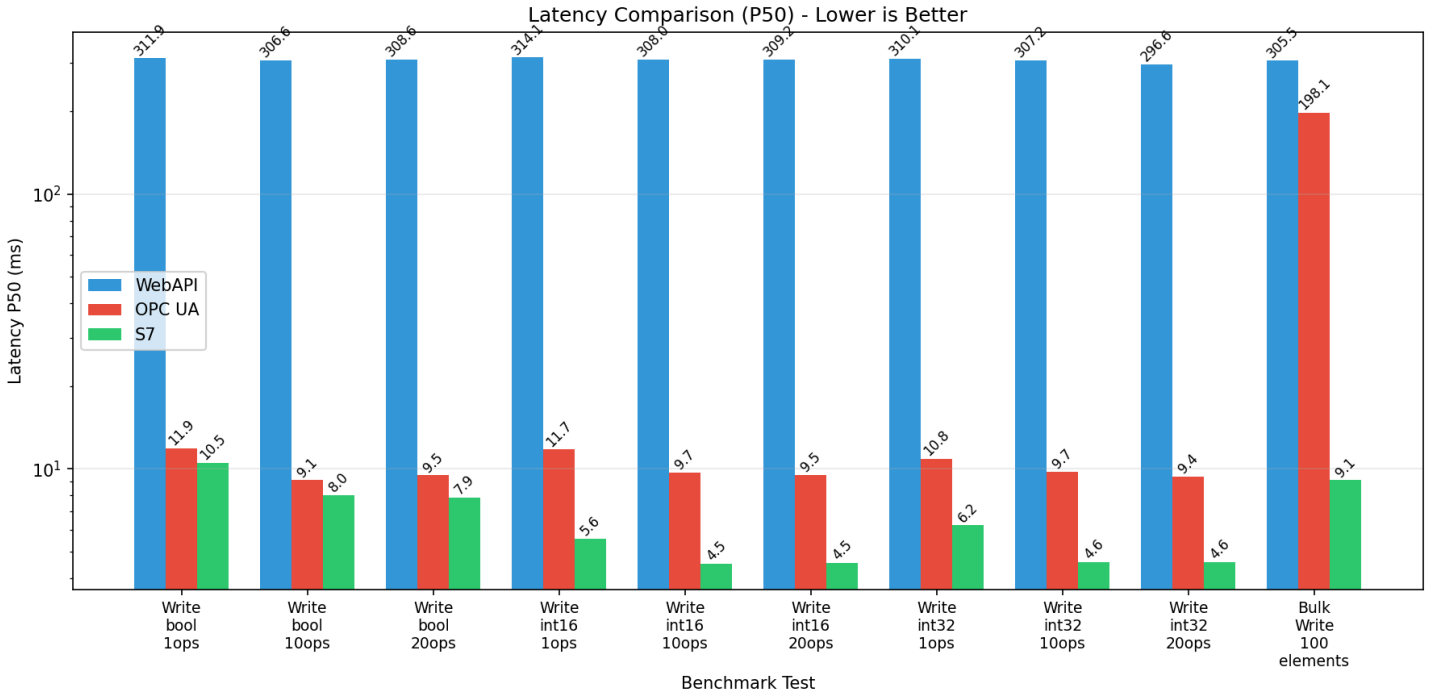
Latenz-Vergleich

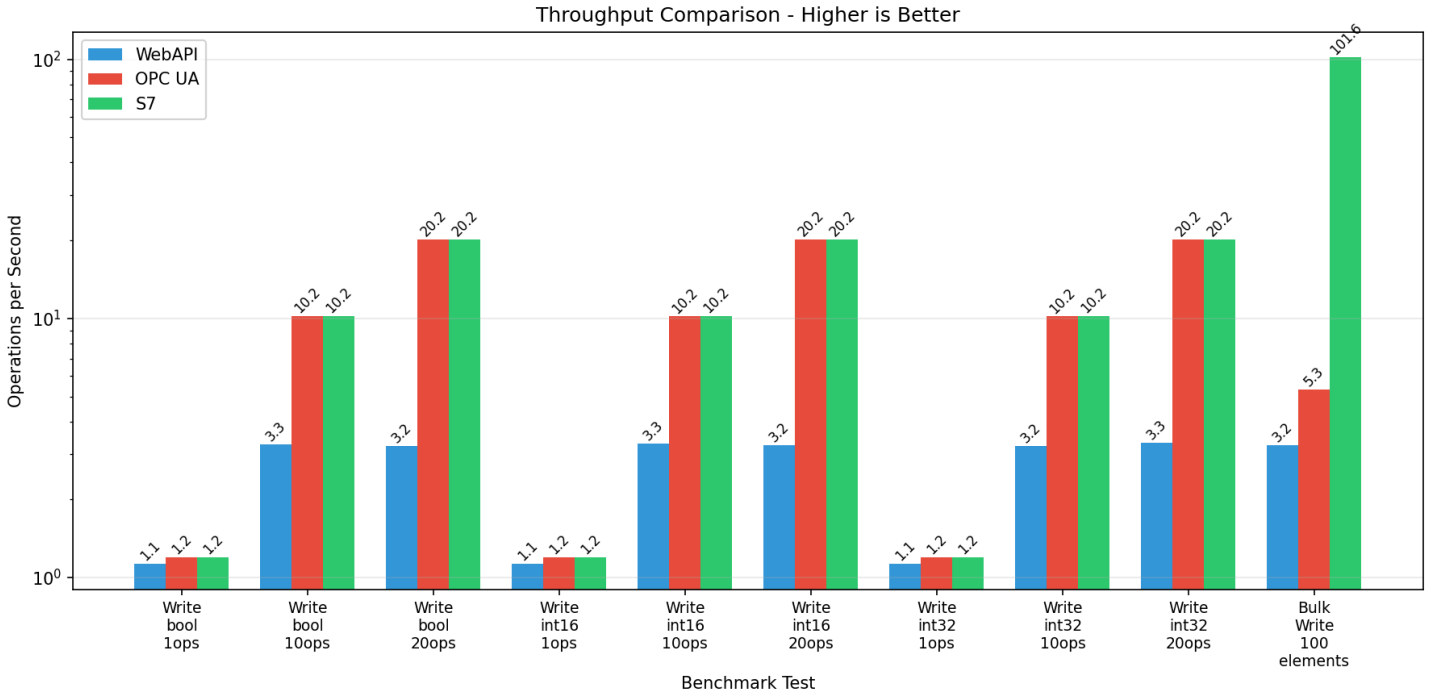
Protokoll	Typische Latenz (P50)	Ursache
WebAPI	~300-310 ms	HTTP-Overhead, SSL-Handshake
OPC UA	~9-12 ms	Subscription-Mechanismus, Protokoll-Overhead
S7	~4-10 ms	Direkter TCP-Socket, minimaler Overhead

Kernaussagen

1. **S7-Protokoll ist durchgehend am schnellsten** – besonders bei Bulk-Writes (~100x schneller als OPC UA)
2. **WebAPI nicht für hochfrequente Zugriffe geeignet** – maximale Rate ~3 ops/s bei 300ms Latenz
3. **OPC UA bietet guten Kompromiss** – nahe an S7-Performance, aber standardisiert und weniger manuell
4. **Datentyp hat kaum Einfluss** – Bool/Int16/Int32 zeigen ähnliche Performance

Grafiken


Latency Comparison (P50) - Lower is Better
Latency P50 (ms)
WebAPI
OPC UA
S7
311.9
11.9
10.5
306.6
9.1
8.0
308.6
9.5
7.9
314.1
11.7
5.6
308.0
9.7
4.5
309.2
9.5
4.5
310.1
10.8
6.2
307.2
9.7
4.6
296.6
9.4
4.6
305.5
198.1
9.1
Write bool 1ops
Write bool 10ops
Write bool 20ops
Write int16 1ops
Write int16 10ops
Write int16 20ops
Write int32 1ops
Write int32 10ops
Write int32 20ops
Bulk Write 100 elements
Benchmark Test


Throughput Comparison - Higher is Better
Operations per Second
WebAPI
OPC UA
S7
1.1
1.2
1.2
3.3
10.2
10.2
3.2
20.2
20.2
1.1
1.2
1.2
3.3
10.2
10.2
3.2
20.2
20.2
1.1
1.2
1.2
3.2
10.2
10.2
3.3
20.2
20.2
3.2
5.3
101.6
Write bool 1ops
Write bool 10ops
Write bool 20ops
Write int16 1ops
Write int16 10ops
Write int16 20ops
Write int32 1ops
Write int32 10ops
Write int32 20ops
Bulk Write 100 elements
Benchmark Test