

Rapport de Projet : Développement d'Applications Réparties

Analyse comparative RPC : XML-RPC, XDR-RPC, JSON-RPC, REST/JSON, gRPC

Module : Développement d'Applications Réparties
Étudiant : Saif Eddine Riahi **enseignant :** Maher Jabberi **Date :** 10 Décembre 2025
Institution : ISSAT Mateur

1. Introduction

Ce projet explore cinq protocoles **Remote Procedure Call (RPC)** : XML-RPC, XDR-RPC, JSON-RPC, REST/JSON et gRPC. Pour chaque protocole, nous avons implémenté un serveur et client Python, capturé le trafic réseau avec Wireshark, puis analysé les différences de sérialisation, taille des trames et lisibilité.

Objectifs : Comparer format sur le fil, lisibilité, sécurité transport et cas d'usage pour chaque protocole RPC.

2. Environnement & Configuration

Technologies

- **Python 3.9+**, Wireshark 4.0+
- **Bibliothèques :** xmlrpc (stdlib), struct, Flask, requests, grpcio, grpcio-tools

Ports utilisés

Protocole	Port	Fonction
XML-RPC	9000	Addition de deux entiers
XDR-RPC	9001	Inversion d'une chaîne
JSON-RPC	9001	Minimum de deux valeurs
REST/JSON	9002	POST /reverse (inversion)
gRPC	50051	Longueur d'une chaîne

3. Protocole XML-RPC

Synthèse

XML-RPC sérialise en XML sur HTTP. Format textuel très lisible mais verbeux.

Implémentation

```
# Serveur
from xmlrpc.server import SimpleXMLRPCServer
def add(x, y): return x + y
server = SimpleXMLRPCServer(('localhost', 9000))
server.register_function(add, 'add')
server.serve_forever()

# Client
import xmlrpc.client
proxy = xmlrpc.client.ServerProxy('http://localhost:9000')
print(proxy.add(5, 3)) # Output: 8
```

Analyse Wireshark

Requête HTTP POST contenant XML :

The image shows a Wireshark packet capture of an HTTP POST request. The top pane shows a list of packets, with packet 34 selected. The middle pane shows the details of packet 34, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The bottom pane shows the raw packet data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
34	2.133008	127.0.0.1	127.0.0.1	HTTP/X...	231	POST /RPC2 HTTP/1.1
36	2.144795	127.0.0.1	127.0.0.1	HTTP/X...	302	HTTP/1.0 200 OK
57	4.183847	127.0.0.1	127.0.0.1	HTTP/X...	234	POST /RPC2 HTTP/1.1
59	4.186420	127.0.0.1	127.0.0.1	HTTP/X...	303	HTTP/1.0 200 OK
84	6.229626	127.0.0.1	127.0.0.1	HTTP/X...	235	POST /RPC2 HTTP/1.1
86	6.230450	127.0.0.1	127.0.0.1	HTTP/X...	304	HTTP/1.0 200 OK

Frame 34: Packet, 231 bytes on wire (1848 bits), 231 bytes captured on interface eth0, from 127.0.0.1 to 127.0.0.1

Ethernet II, Src: VirtualBox (08:00:00:00:00:00), Dst: VirtualBox (08:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 61498, Dst Port: 9000, Seq: 32145, Win: 0, Len: 231

Hypertext Transfer Protocol

POST /RPC2 HTTP/1.1

Host: localhost

User-Agent: Python/2.7.10

Content-Type: text/xml

<?xml version="1.0"?><methodCall><methodName>add</methodName><params><param><value>5</value></param><param><value>3</value></param></params></methodCall>

Observations : Frame ≈231 bytes. Payload XML `<methodCall><methodName>add</methodName>` totalement lisible.

Réponse HTTP 200 OK avec XML :

http && tcp.port == 9000						
No.	Time	Source	Destination	Protocol	Length	Info
34	2.133088	127.0.0.1	127.0.0.1	HTTP/X...	231	POST /RPC2 HTTP/1.1
36	2.144795	127.0.0.1	127.0.0.1	HTTP/X...	302	HTTP/1.0 200 OK
57	4.183847	127.0.0.1	127.0.0.1	HTTP/X...	234	POST /RPC2 HTTP/1.1
59	4.186420	127.0.0.1	127.0.0.1	HTTP/X...	303	HTTP/1.0 200 OK
84	6.229626	127.0.0.1	127.0.0.1	HTTP/X...	235	POST /RPC2 HTTP/1.1
86	6.230450	127.0.0.1	127.0.0.1	HTTP/X...	304	HTTP/1.0 200 OK

<p>▶ Frame 36: Packet, 302 bytes on wire (2416 bits), 302 bytes captured on interface</p> <p>▶ Null/Loopback</p> <p>▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1</p> <p>▶ Transmission Control Protocol, Src Port: 9000, Dst Port: 61498, Seq: 1234567890, Win: 65535, Len: 0</p> <p>▶ Hypertext Transfer Protocol</p> <p>▶ eXtensible Markup Language</p>	<pre> 0000 02 00 00 00 45 00 01 2a 8e cd 40 00 80 06 00 00 E..*..@.... 0010 7f 00 00 01 7f 00 00 01 23 28 f0 3a 87 ee dd 4a #(:...J 0020 09 22 57 fd 50 18 00 fe 9f e8 00 00 48 54 54 50 .."W P... ..HTTP 0030 2f 31 2e 30 20 32 30 30 20 4f 4b 0d 0a 53 65 72 /1.0 200 OK..Ser 0040 76 65 72 3a 20 42 61 73 65 48 54 54 50 2f 30 2e ver: Bas eHTTP/0. 0050 36 20 50 79 74 68 6f 6e 2f 33 2e 31 31 2e 39 0d 6 Python /3.11.9. 0060 0a 44 61 74 65 3a 20 54 75 65 2c 20 31 36 20 44 .Date: T ue, 16 D 0070 65 63 20 32 30 32 35 20 32 30 3a 35 31 3a 31 32 ec 2025 20:51:12 0080 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 74 79 GMT..Co ntent-ty 0090 70 65 3a 20 74 65 78 74 2f 78 6d 6c 0d 0a 43 6f pe: text /xml..Co 00a0 6e 74 65 6e 74 2d 6c 65 6e 67 74 68 3a 20 31 32 ntent-le ngth: 12 00b0 31 0d 0a 0d 0a 3c 3f 78 6d 6c 20 76 65 72 73 69 1...<?x ml versi 00c0 6f 6e 3d 27 31 2e 30 27 3f 3e 0a 3c 6d 65 74 68 on='1.0' ?><meth 00d0 6f 64 52 65 73 70 6f 6e 73 65 3e 0a 3c 70 61 72 odRespon se><par 00e0 61 6d 73 3e 0a 3c 70 61 72 61 6d 3e 0a 3c 76 61 ams><pa ram><va 00f0 6c 75 65 3e 3c 69 6e 74 3e 38 3c 2f 69 6e 74 3e lue><int >8</int> 0100 3c 2f 76 61 6c 75 65 3e 0a 3c 2f 70 61 72 61 6d </value> </param 0110 3e 0a 3c 2f 70 61 72 61 6d 73 3e 0a 3c 2f 6d 65 ></para ms></me 0120 74 68 6f 64 52 65 73 70 6f 6e 73 65 3e 0a thodRespon se> </pre>
---	--

Observations : Frame ≈302 bytes. `<methodResponse><int>8</int>` visible en clair. Verbose XML augmente la taille.

Analyse : Lisibilité ★★★★★, Taille importante, HTTP/HTTPS pour sécurité, idéal pour debugging.

4. Protocole XDR-RPC

Synthèse

XDR encode en binaire (big-endian). Compact mais illisible sans décodeur.

Implémentation

```

# Serveur
import struct
from http.server import HTTPServer, BaseHTTPRequestHandler

class XDRHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        body = self.rfile.read(int(self.headers['Content-Length']))
        str_len, = struct.unpack('!I', body[:4])
        input_str = body[4:4+str_len].decode('utf-8')
        result = input_str[::-1] # Inverse
        response = struct.pack('!I', len(result)) + result.encode()
        self.send_response(200)

```

```

self.end_headers()
self.wfile.write(response)

server = HTTPServer(('localhost', 9001), XDRHandler)
server.serve_forever()

```

Analyse Wireshark

Requête TCP avec payload binaire XDR :

No.	Time	Source	Destination	Protocol	Length	Info
8	0.397623	127.0.0.1	127.0.0.1	TCP	56	65101 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
9	0.397666	127.0.0.1	127.0.0.1	TCP	56	9001 → 65101 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
10	0.397689	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=1 Ack=1 Win=65280 Len=0
11	0.397732	127.0.0.1	127.0.0.1	TCP	56	65101 → 9001 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=12
12	0.397742	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [ACK] Seq=1 Ack=13 Win=65280 Len=0
13	0.397949	127.0.0.1	127.0.0.1	TCP	56	9001 → 65101 [PSH, ACK] Seq=1 Ack=13 Win=65280 Len=12
14	0.397966	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=13 Ack=13 Win=65280 Len=0
15	0.397980	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [FIN, ACK] Seq=13 Ack=13 Win=65280 Len=0
16	0.397988	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=13 Ack=14 Win=65280 Len=0
17	0.398057	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [FIN, ACK] Seq=13 Ack=14 Win=65280 Len=0
18	0.398086	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [ACK] Seq=14 Ack=14 Win=65280 Len=0
19	0.398438	127.0.0.1	127.0.0.1	TCP	56	65102 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
20	0.398481	127.0.0.1	127.0.0.1	TCP	56	9001 → 65102 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
21	0.398500	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [ACK] Seq=1 Ack=1 Win=65280 Len=0
22	0.398522	127.0.0.1	127.0.0.1	TCP	56	65102 → 9001 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=12
23	0.398530	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [ACK] Seq=1 Ack=13 Win=65280 Len=0
24	0.398765	127.0.0.1	127.0.0.1	TCP	56	9001 → 65102 [PSH, ACK] Seq=1 Ack=13 Win=65280 Len=12
25	0.398777	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [ACK] Seq=13 Ack=13 Win=65280 Len=0
26	0.398788	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [FIN, ACK] Seq=13 Ack=13 Win=65280 Len=0
27	0.398805	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [FIN, ACK] Seq=13 Ack=14 Win=65280 Len=0
28	0.398813	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [ACK] Seq=14 Ack=14 Win=65280 Len=0
29	0.399082	127.0.0.1	127.0.0.1	TCP	56	65104 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

▶ Frame 11: Packet, 56 bytes on wire (448 bits), 56 bytes captured	0000	02 00 00 00 45 00 00 34	93 15 40 00 80 06 00 00E..4..@....
▶ Null/Loopback	0010	7f 00 00 01 7f 00 00 01	fe 4d 23 29 16 01 f3 06(M#)....
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	71 6e 3b 81 50 18 00 ff	95 79 00 00 00 00 00 05	qn;P...y.....
▶ Transmission Control Protocol, Src Port: 65101, Dst Port: 9001, S	0030	68 65 6c 6c 6f 00 00 00		hello...
▶ Data (12 bytes)				

Observations : Frame ≈56 bytes. Data (12 bytes) en binaire, visible seulement en hex dump.

Réponse TCP avec XDR binaire :

No.	Time	Source	Destination	Protocol	Length	Info
8	0.397623	127.0.0.1	127.0.0.1	TCP	56	65101 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
9	0.397666	127.0.0.1	127.0.0.1	TCP	56	9001 → 65101 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
10	0.397732	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=1 Ack=1 Win=65280 Len=0
11	0.397732	127.0.0.1	127.0.0.1	TCP	56	65101 → 9001 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=12
12	0.397742	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [ACK] Seq=1 Ack=13 Win=65280 Len=0
13	0.397949	127.0.0.1	127.0.0.1	TCP	56	9001 → 65101 [PSH, ACK] Seq=1 Ack=13 Win=65280 Len=12
14	0.397966	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=13 Ack=13 Win=65280 Len=0
15	0.397980	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [FIN, ACK] Seq=13 Ack=13 Win=65280 Len=0
16	0.397988	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [ACK] Seq=13 Ack=14 Win=65280 Len=0
17	0.398057	127.0.0.1	127.0.0.1	TCP	44	65101 → 9001 [FIN, ACK] Seq=13 Ack=14 Win=65280 Len=0
18	0.398086	127.0.0.1	127.0.0.1	TCP	44	9001 → 65101 [ACK] Seq=14 Ack=14 Win=65280 Len=0
19	0.398438	127.0.0.1	127.0.0.1	TCP	56	65102 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
20	0.398481	127.0.0.1	127.0.0.1	TCP	56	9001 → 65102 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
21	0.398500	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [ACK] Seq=1 Ack=1 Win=65280 Len=0
22	0.398522	127.0.0.1	127.0.0.1	TCP	56	65102 → 9001 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=12
23	0.398530	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [ACK] Seq=1 Ack=13 Win=65280 Len=0
24	0.398765	127.0.0.1	127.0.0.1	TCP	56	9001 → 65102 [PSH, ACK] Seq=1 Ack=13 Win=65280 Len=12
25	0.398777	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [ACK] Seq=13 Ack=13 Win=65280 Len=0
26	0.398788	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [FIN, ACK] Seq=13 Ack=13 Win=65280 Len=0
27	0.398805	127.0.0.1	127.0.0.1	TCP	44	65102 → 9001 [FIN, ACK] Seq=13 Ack=14 Win=65280 Len=0
28	0.398813	127.0.0.1	127.0.0.1	TCP	44	9001 → 65102 [ACK] Seq=14 Ack=14 Win=65280 Len=0
29	0.399082	127.0.0.1	127.0.0.1	TCP	56	65104 → 9001 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM

▶ Frame 13: Packet, 56 bytes on wire (448 bits), 56 bytes captured	0000	02 00 00 00 45 00 00 34	93 17 40 00 80 06 00 00E..4..@....
▶ Null/Loopback	0010	7f 00 00 01 7f 00 00 01	23 29 fe 4d 71 6e 3b 81(M#)Mqn;
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	16 01 f3 12 50 18 00 ff	95 6d 00 00 00 00 00 05P...m.....
▶ Transmission Control Protocol, Src Port: 9001, Dst Port: 65101, S	0030	6f 6c 6c 65 68 00 00 00		olleh...
▶ Data (12 bytes)				

Observations : Frame ≈56 bytes. Chaîne "olleh" encodée en binaire compact.

Analyse : Lisibilité ★☆☆☆☆, Très compact (~75% plus petit qu'XML), difficile à analyser manuellement.

5. Protocole JSON-RPC

Synthèse

JSON-RPC 2.0 : format structuré JSON avec `jsonrpc`, `method`, `params`, `id`.

Implémentation

```
# Serveur
from http.server import HTTPServer, BaseHTTPRequestHandler
import json

class JSONRPCHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        body = json.loads(self.rfile.read(int(self.headers['Content-Length'])))
        if body.get('method') == 'min':
            result = min(body.get('params', []))
            response = {"jsonrpc": "2.0", "result": result, "id": body.get('id')}
        self.send_response(200)
        self.end_headers()
        self.wfile.write(json.dumps(response).encode())

server = HTTPServer(('localhost', 9001), JSONRPCHandler)
server.serve_forever()
```

Analyse Wireshark

Requête HTTP avec JSON-RPC 2.0 :

http && tcp.port == 9001					
No.	Time	Source	Destination	Protocol	Length Info
14	1.532341	127.0.0.1	127.0.0.1	HTTP/J...	106 POST / HTTP/1.1 , JSON (application/json)
20	1.533449	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
57	13.413667	127.0.0.1	127.0.0.1	HTTP/J...	107 POST / HTTP/1.1 , JSON (application/json)
63	13.414930	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
86	15.471480	127.0.0.1	127.0.0.1	HTTP/J...	107 POST / HTTP/1.1 , JSON (application/json)
92	15.472595	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
118	17.521939	127.0.0.1	127.0.0.1	HTTP/J...	106 POST / HTTP/1.1 , JSON (application/json)
124	17.527937	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)

▶ Frame 86: Packet, 107 bytes on wire (856 bits), 107 bytes capture

▶ Null/Loopback

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 57665, Dst Port: 9001, S

▶ [2 Reassembled TCP Segments (261 bytes): #84(198), #86(63)]

▶ Hypertext Transfer Protocol

▶ JavaScript Object Notation: application/json

0000 02 00 00 00 45 00 00 67 92 51 40 00 80 06 00 00E..g..Q@....

0010 7f 00 00 01 7f 00 00 01 e1 41 23 29 aa 2e a6 50A#)..P

0020 96 53 36 c3 50 18 00 ff 14 a7 00 00 7b 22 6a 73 ..S6.P....{"js

0030 6f 6e 72 70 63 22 3a 20 22 32 2e 30 22 2c 20 22 onrpc": "2.0", "

0040 6d 65 74 68 6f 64 22 3a 20 22 6d 69 6e 22 2c 20 method": "min",

0050 22 70 61 72 61 6d 73 22 3a 20 5b 2d 33 2c 20 37 "params": [-3, 7

0060 5d 2c 20 22 69 64 22 3a 20 31 7d], "id": 1}

Observations : Frame ≈107 bytes. JSON {"jsonrpc":"2.0","method":"min","params":[-3,7]} lisible.

Réponse HTTP avec result JSON :

http && tcp.port == 9001					
No.	Time	Source	Destination	Protocol	Length Info
14	1.532341	127.0.0.1	127.0.0.1	HTTP/J...	106 POST / HTTP/1.1 , JSON (application/json)
20	1.533449	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
57	13.413667	127.0.0.1	127.0.0.1	HTTP/J...	107 POST / HTTP/1.1 , JSON (application/json)
63	13.414930	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
86	15.471480	127.0.0.1	127.0.0.1	HTTP/J...	107 POST / HTTP/1.1 , JSON (application/json)
92	15.472595	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)
118	17.521939	127.0.0.1	127.0.0.1	HTTP/J...	106 POST / HTTP/1.1 , JSON (application/json)
124	17.527937	127.0.0.1	127.0.0.1	HTTP/J...	44 HTTP/1.0 200 OK , JSON (application/json)

▶ Frame 92: Packet, 44 bytes on wire (352 bits), 44 bytes captured

▶ Null/Loopback

▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

▶ Transmission Control Protocol, Src Port: 9001, Dst Port: 57665, S

▶ [3 Reassembled TCP Segments (165 bytes): #88(124), #90(41), #92(0

▶ Hypertext Transfer Protocol

▶ JavaScript Object Notation: application/json

0000 02 00 00 00 45 00 00 28 92 57 40 00 80 06 00 00E..(..W@....

0010 7f 00 00 01 7f 00 00 01 23 29 e1 41 96 53 37 68A..S7h

0020 aa 2e a6 8f 50 11 00 fe 8d ee 00 00P.....

Observations : Petite frame. JSON {"result":-3,"id":1} clair et structuré.

Analyse : Lisibilité ★★★★★, Taille intermédiaire, bon compromis texte/compacité.

6. Protocole REST/JSON

Synthèse

Architecture REST : ressources HTTP + verbes (POST, GET). État échangé en JSON.

Implémentation

```
# Serveur Flask
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/reverse', methods=['POST'])
def reverse():
    data = request.get_json()
    return jsonify({'result': data['value'][::-1]}), 200

app.run(host='localhost', port=9002)
```

Analyse Wireshark

Requête POST /reverse avec JSON :

No.	Time	Source	Destination	Protocol	Length	Info
230	157.934011	127.0.0.1	127.0.0.1	HTTP/J...	62	POST /reverse HTTP/1.1 , JSON (application/json)
234	157.940272	127.0.0.1	127.0.0.1	HTTP/J...	63	HTTP/1.1 200 OK , JSON (application/json)
261	159.992121	127.0.0.1	127.0.0.1	HTTP/J...	63	POST /reverse HTTP/1.1 , JSON (application/json)
265	159.997520	127.0.0.1	127.0.0.1	HTTP/J...	64	HTTP/1.1 200 OK , JSON (application/json)
292	162.058070	127.0.0.1	127.0.0.1	HTTP/J...	63	POST /reverse HTTP/1.1 , JSON (application/json)
296	162.063308	127.0.0.1	127.0.0.1	HTTP/J...	64	HTTP/1.1 200 OK , JSON (application/json)
317	164.101718	127.0.0.1	127.0.0.1	HTTP/J...	57	POST /reverse HTTP/1.1 , JSON (application/json)
321	164.105577	127.0.0.1	127.0.0.1	HTTP/J...	58	HTTP/1.1 200 OK , JSON (application/json)

▶ Frame 230: Packet, 62 bytes on wire (496 bits), 62 bytes captured	0000	02 00 00 00 45 00 00 3a	94 fe 40 00 80 06 00 00E...: @.....
▶ Null/Loopback	0010	7f 00 00 01 7f 00 00 01	c5 f3 23 2a 4c 14 9b 57L..W
▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0020	d4 4b ba 89 50 18 00 ff	3d 5d 00 00 7b 22 76 61	..K..P...=]...{"va
▶ Transmission Control Protocol, Src Port: 50675, Dst Port: 9002, S...	0030	6c 75 65 22 3a 20 22 68	65 6c 6c 6f 22 7d	lue": "h ello"}
▶ [2 Reassembled TCP Segments (223 bytes): #228(205), #230(18)]				
▶ Hypertext Transfer Protocol				
▶ JavaScript Object Notation: application/json				

Observations : Frame ≈62 bytes. URI `/reverse` + JSON `{"value":"hello"}` explicite.

Réponse HTTP 200 OK avec JSON :

http && tcp.port == 9002					
No.	Time	Source	Destination	Protocol	Length Info
230	157.934011	127.0.0.1	127.0.0.1	HTTP/J...	62 POST /reverse HTTP/1.1 , JSON (application/json)
234	157.940272	127.0.0.1	127.0.0.1	HTTP/J...	63 HTTP/1.1 200 OK , JSON (application/json)
261	159.992121	127.0.0.1	127.0.0.1	HTTP/J...	63 POST /reverse HTTP/1.1 , JSON (application/json)
265	159.997520	127.0.0.1	127.0.0.1	HTTP/J...	64 HTTP/1.1 200 OK , JSON (application/json)
292	162.058070	127.0.0.1	127.0.0.1	HTTP/J...	63 POST /reverse HTTP/1.1 , JSON (application/json)
296	162.063308	127.0.0.1	127.0.0.1	HTTP/J...	64 HTTP/1.1 200 OK , JSON (application/json)
317	164.101718	127.0.0.1	127.0.0.1	HTTP/J...	57 POST /reverse HTTP/1.1 , JSON (application/json)
321	164.105577	127.0.0.1	127.0.0.1	HTTP/J...	58 HTTP/1.1 200 OK , JSON (application/json)

<p>Frame 234: Packet, 63 bytes on wire (504 bits), 63 bytes captured</p> <p>Null/Loopback</p> <p>Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1</p> <p>Transmission Control Protocol, Src Port: 9002, Dst Port: 50675, S</p> <p>[2 Reassembled TCP Segments (184 bytes): #232(165), #234(19)]</p> <p>Hypertext Transfer Protocol</p> <p>JavaScript Object Notation: application/json</p>	<pre> 0000 02 00 00 00 45 00 00 3b 95 02 40 00 00 06 00 00 E.;..@.... 0010 7f 00 00 01 7f 00 00 01 23 2a c5 f3 d4 4b bb 2e *...K. 0020 4c 14 9b 69 50 18 00 ff 40 35 00 00 7b 22 72 65 L..iP...@5...{"re 0030 73 75 6c 74 22 3a 22 6f 6c 6c 65 68 22 7d 0a sult":"o lleh"} </pre>
---	---

Observations : Frame ≈63 bytes. `{"result":"olleh"}` compact et clair.

Analyse : Lisibilité ★★★★★, Très compact, HTTP standard universel, idéal pour APIs web.

7. Protocole gRPC

Synthèse

gRPC : Protocol Buffers (binaire) + HTTP/2. Haute performance, streaming natif.

Implémentation

```
// length.proto
syntax = "proto3";
package length;

service LengthService {
  rpc GetLength(StringRequest) returns (StringReply) {}
}

message StringRequest { string value = 1; }
message StringReply { int32 length = 1; }
```

```
# Serveur
import grpc, length_pb2, length_pb2_grpc
from concurrent import futures

class LengthServicer(length_pb2_grpc.LengthServiceServicer):
    def GetLength(self, request, context):
```

```

return length_pb2.StringReply(length=len(request.value))

server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
length_pb2_grpc.add_LengthServiceServicer_to_server(LengthServicer(), server)
server.add_insecure_port('[::]:50051')
server.start()
server.wait_for_termination()

```

Analyse Wireshark

Requête HTTP/2 avec Protocol Buffers :

Wireshark capture of an HTTP/2 POST request to /length.LengthService/GetLength. The packet list shows a GRPC message of 342 bytes. The packet details show the GRPC message structure with headers and data. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
8	0.135403	:::1	:::1	HTTP2	146	Magic, SETTINGS[0], WINDOW_UPDATE[0]
10	0.135439	:::1	:::1	HTTP2	110	SETTINGS[0], WINDOW_UPDATE[0]
12	0.135510	:::1	:::1	HTTP2	73	SETTINGS[0]
14	0.135531	:::1	:::1	HTTP2	73	SETTINGS[0]
16	0.135610	:::1	:::1	GRPC/P...	342	HEADERS[1]: POST /length.LengthService/GetLength, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
18	0.135710	:::1	:::1	HTTP2	81	PING[0]
20	0.135732	:::1	:::1	HTTP2	81	PING[0]
22	0.136144	:::1	:::1	GRPC/P...	219	HEADERS[1]: 200 OK, DATA[1] (GRPC) (PROTOBUF), HEADERS[1], WINDOW_UPDATE[0]
24	0.136226	:::1	:::1	HTTP2	81	PING[0]
26	0.136267	:::1	:::1	HTTP2	81	PING[0]
35	0.137055	:::1	:::1	HTTP2	146	Magic, SETTINGS[0], WINDOW_UPDATE[0]
37	0.137081	:::1	:::1	HTTP2	110	SETTINGS[0], WINDOW_UPDATE[0]
39	0.137121	:::1	:::1	HTTP2	73	SETTINGS[0]
41	0.137143	:::1	:::1	HTTP2	73	SETTINGS[0]
43	0.137228	:::1	:::1	GRPC/P...	348	HEADERS[1]: POST /length.LengthService/GetLength, WINDOW_UPDATE[1], DATA[1] (GRPC) (PROTOBUF), WINDOW_UPDATE[0]
45	0.137293	:::1	:::1	HTTP2	81	PING[0]
47	0.137316	:::1	:::1	HTTP2	81	PING[0]
49	0.137554	:::1	:::1	GRPC/P...	219	HEADERS[1]: 200 OK, DATA[1] (GRPC) (PROTOBUF), HEADERS[1], WINDOW_UPDATE[0]
51	0.137622	:::1	:::1	HTTP2	81	PING[0]
53	0.137664	:::1	:::1	HTTP2	81	PING[0]
62	0.138348	:::1	:::1	HTTP2	146	Magic, SETTINGS[0], WINDOW_UPDATE[0]
64	0.138389	:::1	:::1	HTTP2	110	SETTINGS[0], WINDOW_UPDATE[0]

Frame 16: Packet, 342 bytes on wire (2736 bits), 342 bytes captured on interface eth0, 342 bytes from :::1 to :::1

Null/Loopback

Internet Protocol Version 6, Src: :::1, Dst: :::1

Transmission Control Protocol, Src Port: 59994, Dst Port: 50051

Hypertext Transfer Protocol 2

Hypertext Transfer Protocol 2

GRPC Message: /length.LengthService/GetLength, Request

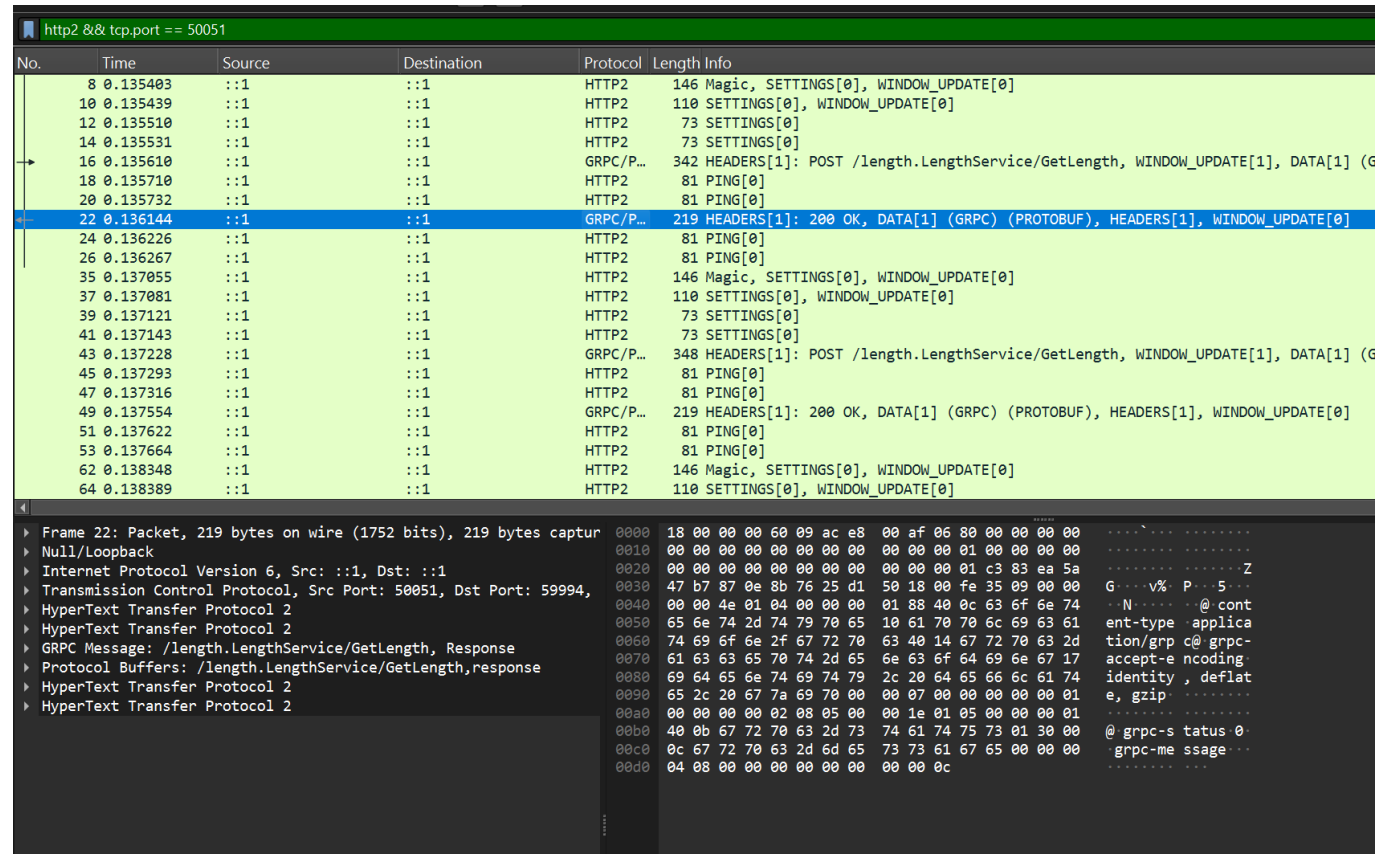
Protocol Buffers: /length.LengthService/GetLength,request

Hypertext Transfer Protocol 2

0000 18 00 00 00 60 01 f9 a4 01 2a 06 00 00 00 00 00*.....
 0010 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00Z..
 0020 00 00 00 00 00 00 00 00 00 00 00 01 ea 5a c3 83v\$G...P..
 0030 8b 75 24 ea 47 b7 85 fd 59 18 00 ff c1 85 00 00@:path
 0040 00 00 de 01 04 00 00 00 01 40 95 3a 70 61 74 68 .../length.LengthS
 0050 1f 2f 6c 65 6e 67 74 68 2e 4c 65 6e 67 74 68 53 ...ervice/GetLength
 0060 65 72 76 69 63 65 2f 47 65 74 4c 65 6e 67 74 68 @:authority loc
 0070 40 0a 3a 61 75 74 68 6f 72 69 74 79 0f 6c 6f 63 @:localhost:50051 @
 0080 61 6c 68 6f 73 74 3a 35 30 30 35 31 83 86 40 0c content-type app
 0090 63 6f 6e 74 65 6e 74 2d 74 79 70 65 10 61 70 70 lication /grpc@t
 00a0 6c 69 63 61 74 69 6f 6e 2f 67 72 70 63 40 02 74 e-trails@grpc
 00b0 65 08 74 72 61 69 6c 65 72 73 40 14 67 72 70 63 -accept-encoding
 00c0 2d 61 63 63 65 70 74 2d 65 6e 63 6f 64 69 6e 67 identity, defla
 00d0 17 69 64 65 6e 74 69 74 79 2c 20 64 65 66 6c 61 te, gzip@user-a
 00e0 74 65 2c 20 67 7a 69 70 40 0a 75 73 65 72 2d 61 gent2grp c-python
 00f0 67 65 6e 74 32 67 72 70 63 2d 70 79 74 68 6f 6e /1.76.0 grpc-c/5
 0100 2f 31 2e 37 36 2e 30 20 67 72 70 63 2d 63 2f 35 1.0.0 (w
 0110 31 2a 30 2e 30 20 28 77 69 6e 64 6f 77 73 3b 20 indows;
 0120 63 68 74 74 70 32 29 00 00 04 08 00 00 00 00 01 chhttp2)
 0130 00 00 00 05 00 00 0c 00 01 00 00 00 01 00 00 00hell o.....
 0140 00 07 0a 05 68 65 6c 6c 6f 00 00 04 08 00 00 00hell o.....
 0150 00 00 00 00 00 05

Observations : Frame ≈342 bytes. HTTP/2 POST /length.LengthService/GetLength, DATA protobuf binaire illisible.

Réponse HTTP/2 avec protobuf :



Observations : Frame ≈219 bytes. Réponse binaire compacte, nécessite .proto pour interpréter.

Analyse : Lisibilité ★☆☆☆☆, Très compact, HTTP/2 performant, idéal pour microservices.

8. Analyse Comparative

Tableau de synthèse

Critère	XML-RPC	XDR-RPC	JSON-RPC	REST/JSON	gRPC
Lisibilité	★★★★★	★☆☆☆☆	★★★★☆	★★★★★	★☆☆☆☆
Taille (bytes)	230-300	50-100	100-150	60-100	100-350
Format	HTTP+XML	TCP+XDR binaire	HTTP+JSON	HTTP+JSON	HTTP/2+Protobuf
Transport sécurisé	HTTPS	HTTPS	HTTPS	HTTPS	TLS/mTLS
Multi-langage	Excellent	Bon	Excellent	Universel	Excellent
Streaming	Non	Non	Non	Non	Oui (natif)
Cas d'usage	Legacy web	Contraintes bande passante	APIs simples	APIs web modernes	Microservices haute perf

Recommandations par cas d'usage

- **API web publique** → REST/JSON : standard, caching HTTP, universellement supporté

- **Microservices internes** → gRPC : performance, streaming, type-safe
- **Prototype rapide** → REST/JSON ou JSON-RPC : setup minimal, testable immédiatement
- **Système legacy** → XML-RPC ou XDR-RPC : compatibilité existante
- **IoT/embedded** → JSON-RPC : léger, simple, support universel

9. Méthodologie Wireshark

Procédure de capture

1. Ouvrir Wireshark, interface Loopback
2. Filtre par port : `tcp.port == 9000` (XML-RPC), `tcp.port == 9001` (XDR/JSON-RPC), `tcp.port == 9002` (REST), `tcp.port == 50051` (gRPC)
3. Lancer serveur, puis client
4. Enregistrer capture : `File > Export Specified Packets → .pcapng`
5. Stocker dans `wireshark-captures/`

Tests réalisés

Protocole	Fonction	Input	Output attendu	Status
XML-RPC	add	(5, 3)	8	✓ PASS
XDR-RPC	reverse	"hello"	"olleh"	✓ PASS
JSON-RPC	min	[-3, 7]	-3	✓ PASS
REST/JSON	/reverse	{"value":"hello"}	{"result":"olleh"}	✓ PASS
gRPC	GetLength	"hello"	5	✓ PASS

Structure du dépôt Git

```
Projet-RPC/  
├── xml-rpc/  
│   ├── server.py  
│   └── client.py  
├── xdr-rpc/  
│   ├── server.py  
│   └── client.py  
├── json-rpc/  
│   ├── server.py  
│   └── client.py  
├── rest-json/  
│   ├── server.py  
│   └── client.py  
├── grpc/  
│   ├── length.proto  
│   ├── server.py  
│   └── client.py  
└── wireshark-captures/  
    └── xml-rpc-request.png
```

- └─ xml-rpc-response.png
- └─ xdr-rpc-request.png
- └─ xdr-rpc-response.png
- └─ json-rpc-request.png
- └─ json-rpc-response.png
- └─ rest-json-request.png
- └─ rest-json-response.png
- └─ grpc-request.png
- └─ grpc-response.png
- └─ rapport_final.md

10. Conclusion

Ce projet a démontré les différences fondamentales entre cinq approches RPC. **XML-RPC** et **XDR-RPC** sont historiques mais dépassés. **JSON-RPC** offre un bon compromis. **REST/JSON** domine les APIs web modernes grâce à sa simplicité et son universalité. **gRPC** s'impose pour les microservices nécessitant haute performance et streaming.

Les captures Wireshark ont validé les caractéristiques théoriques : XML/JSON sont lisibles mais plus volumineux, XDR/Protobuf sont compacts mais opaques, HTTP/2 optimise le transport.

Perspectives

- Implémenter sécurité (TLS/mTLS, OAuth2)
- Benchmark performance (latence, throughput)
- Containerisation Docker + orchestration
- Monitoring avec Prometheus/Jaeger

Auteur : Saif Eddine Riahi **Dépôt Git :** [GitHub Saif Riahi](#) **Date :** 10 Décembre 2025