Vsomeip

1. SOME/IP introduction
   - ✓ What & Why ?
   - ✓ Comparison
2. SOME/IP run over Ethernet
   - ✓ Ethernet automotive
   - ✓ Unicast
   - ✓ Mutilcast
3. Vsomeip (an implementation of the SOME/IP protocol)
   - ✓ SOME/IP Message Format
   - ✓ Service-Discovery
   - ✓ Request/Response/pub-sub
4. Vsomeip + CommonApi
   - ✓ RPC (Remote procedure call)
5. Demo & Debugging

   https://github.com/minhthedt/vsomeip-example
6. Q&A

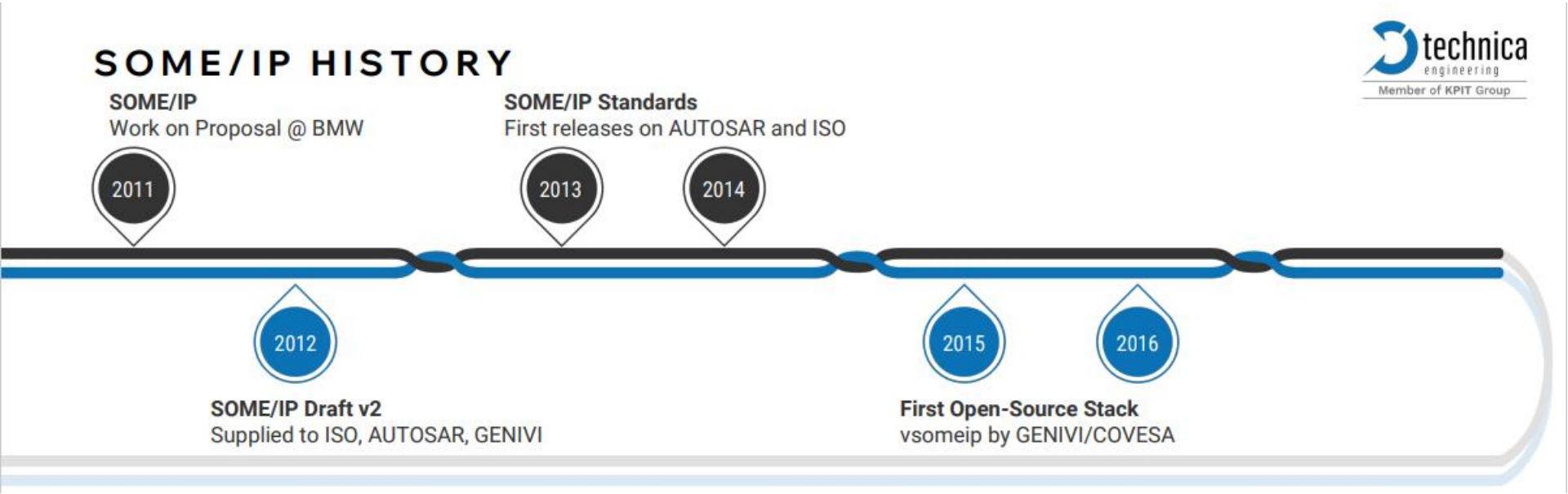
Ha Noi 2025-10-03

1. SOME/IP introduction : What ?

SOME/IP (Scalable service-Oriented Middleware over IP) is a middleware protocol designed for service-oriented communication **over IP networks**. (https://some-ip.com/). It was designed from beginning on to fit devices of different sizes and different operating systems perfectly. This includes small devices like cameras, AUTOSAR devices, and up to head units or **telematics devices**

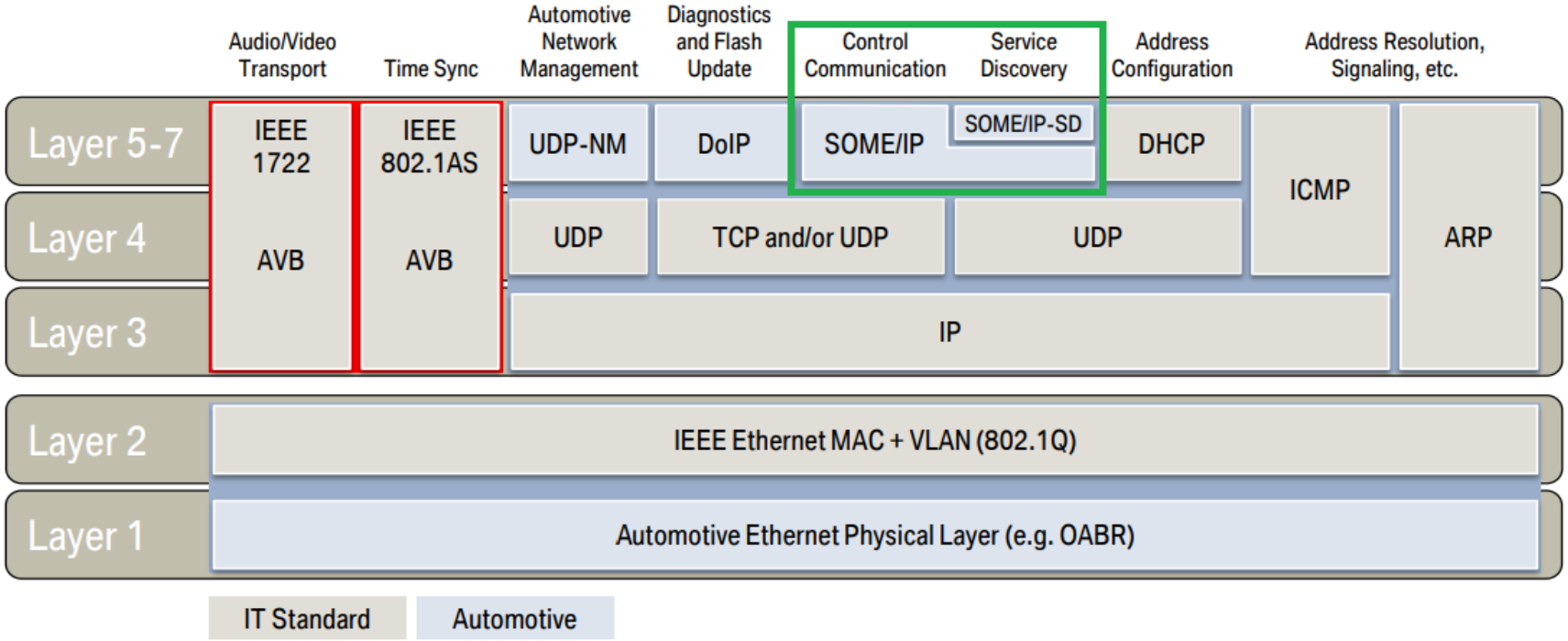
SOME/IP remained as a **proprietary protocol**, originally developed and published by **BMW Group** for internal use and later shared publicly.

vsomeip is an open-source **implementation** of the SOME/IP protocol developed by **Vector Informatik** **(https://github.com/COVESA/vsomeip)**



1. SOME/IP introduction : What ?

OSI Layer	SOME/IP Role
Layer 7 – Application	Defines the **services**, **methods**, and **data structures** used in communication (e.g., RPC calls, service interfaces).
Layer 6 – Presentation	Handles **data encoding/decoding**, **serialization**, and **marshalling** of complex data types (e.g., using CDR or custom formats).
Layer 5 – Session	Manages **service discovery**, **session establishment**, and **lifecycle management** of services (via SOME/IP-SD).



Most parts are reused but on Layer 1 and Layer 7 specific protocols are needed.

1. SOME/IP introduction : Why ?

➢ ECU communication protocols in automotive

In modern automotive systems, the most commonly used CAN, LIN, Ethernet, why ?

real-time control, robust
Up to **1 Mbps**

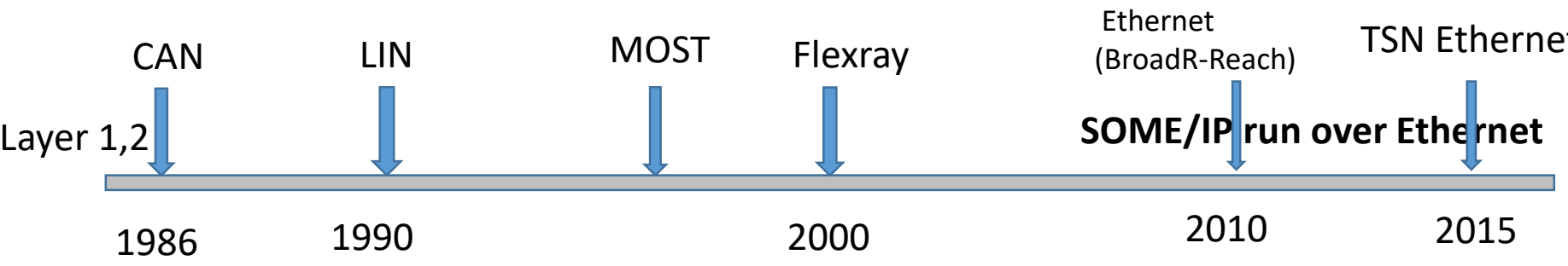
Engine Control Module (ECM)
Transmission Control Module (TCM)
Brake Control Module (ABS/ESC)
Airbag Control Unit
Body Control Module (BCM)
Steering Control Module …

infotainment and multimedia
Up to **150 Mbps**

Infotainment Head Unit
Audio Amplifier ECU
Rear Seat Entertainment ECU
Navigation ECU
Display Control ECU …

high-bandwidth, scalable communication
in modern architectures
Up to **100 Mbps – 10Gbps**

Diagnostic tool
ADAS ECU
Camera ECU
Radar/LiDAR ECU
Telematics ECU (OTA updates, cloud services) …



Window Lift ECU
Seat Adjustment ECU
Mirror Control ECU
Sunroof ECU …

low-speed, low-cost
Up to **20 Kbps**

Brake-by-Wire ECU
Steer-by-Wire ECU
Chassis Control ECU …

safety-critical and
time-deterministic
Up to **10 Mbps**

Autonomous driving domain controller
Sensor fusion ECU (LiDAR, radar, camera)
Brake-by-wire ECU …

Real-time, deterministic,
safety-critical, high-speed

➢ SOME/IP was developed because of:

- Ethernet Adoption in Automotive
- Need for Service-Oriented Architecture (SOA)

1. SOME/IP introduction : Why ?

➢ Ethernet-Based Protocol Comparison

Protocol (layer 5~7)	Architecture	Real-Time Support	Service Discovery	Use Case
SOME/IP (a lightweight binary protocol over TCP/UDP)	Service-Oriented (SOA) RPC (Remote Procedure Call)	⚠ Limited (via TSN)	✓ SOME/IP-SD	ECU-to-ECU communication, **AUTOSAR Adaptive**, infotainment, ADAS
DDS (Data Distribution Service)	Publish-Subscribe	✓ Strong (QoS)	✓ Dynamic	ECU-to-ECU communication, Autonomous driving, robotics, safety-critical systems
gRPC (built on HTTP/2 + Protocol Buffers)	Service-Oriented (SOA RPC (Remote Procedure Call)	⚠Limited	✗ Manual or via registry	Vehicle to backend services, Cloud-native services, infotainment
MQTT (Message Queuing Telemetry Transport)	Publish-Subscribe	✗ Weak	✗ Broker-based	Vehicle-to-cloud telemetry, diagnostics
REST over HTTP	Request-Response	✗ Not real-time	✗ Manual	Vehicle to Web services, connected car platforms

- Both **gRPC** and **SOME/IP** follow a **Service-Oriented Architecture (SOA)** model, but their design goals and environments are quite different. **gRPC** is less performant and heavier for embedded ECUs, ideal for backend/cloud communication

- **DDS** offers better real-time performance than SOME/IP, but SOME/IP is more lightweight and better suited for AUTOSAR

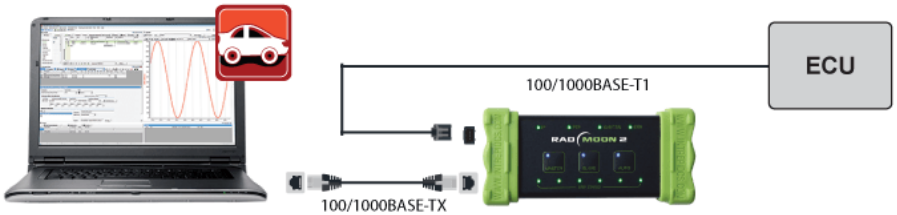
2. SOME/IP run over Ethernet

➢ **Automotive Ethernet (**BroadR-Reach)

BroadR-Reach is a specialized Ethernet physical layer standard developed for automotive applications.
It was originally created by Broadcom

Feature	Traditional Ethernet	BroadR-Reach
Cable Type	Rj45(8 wires) 	Single twisted pair (2 wires) 
Cable Length	Up to 100 meters	Up to 15 meters (UTP), 40m (STP)
Duplex Mode	Full-duplex	Full-duplex
Weight & Cost	Heavier and more expensive	Lighter and cheaper

➢ **Convert Automotive Ethernet <-> Traditional Ethernet**: Media Converter



2. SOME/IP run over Ethernet

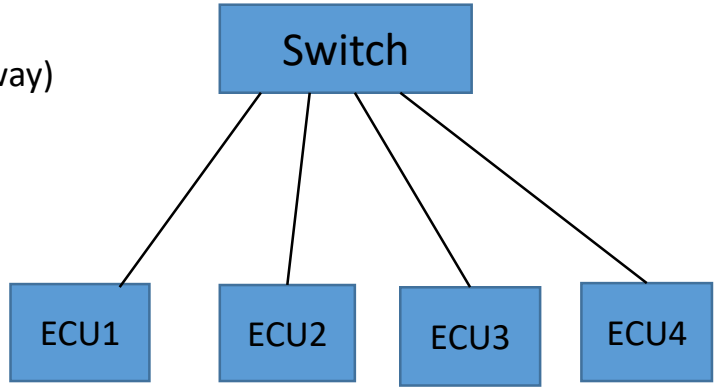
➢ Automotive Ethernet: Topology

❑ **Daisy chain topology** (less common): is often used in cost-sensitive or space-constrained design.
Disadvantages : If one ECU or link fails → breaks the chain, harder fault isolation and timing control

❑ **Star topology** (**most common today**)**:** used in most current automotive Ethernet architectures

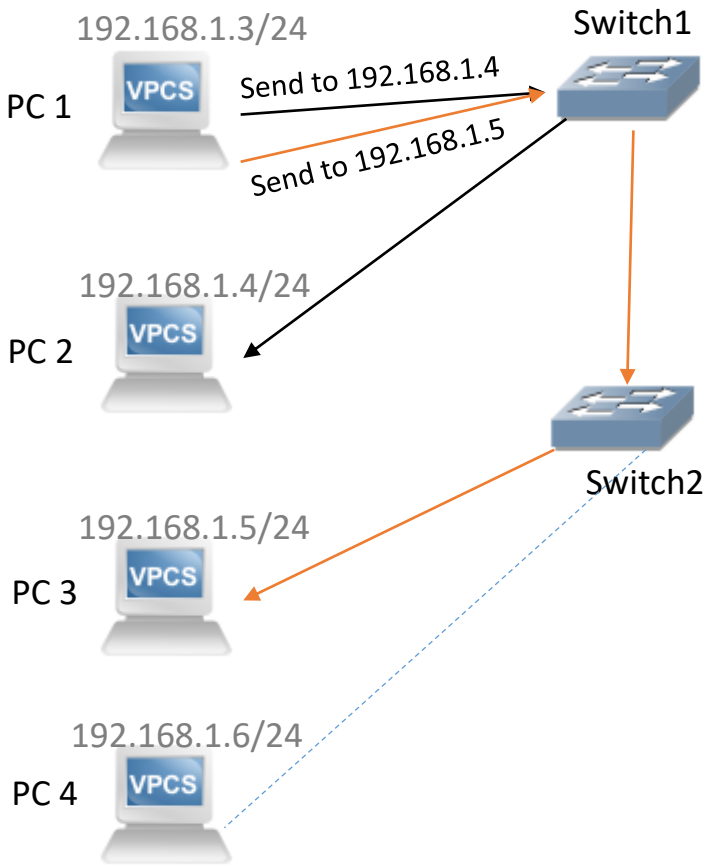
Advantages:
Simple management
Easier diagnostics (one link failure doesn't affect others)
Supports mixed networks (Ethernet ↔ CAN, LIN, FlexRay via gateway)

Disadvantages:
Requires more cabling to the central switch
Slightly higher cost due to extra switch port count



2. SOME/IP run over Ethernet

➢ Unicast (TCP/UDP)



If **PC1** wants to send the same data to **PC2** and **PC3**, it must send two separate packets:

PC1 → PC2 : Packet #1

PC1 → PC3 : Packet #2

➞ This means:

2 copies of the same data are transmitted.

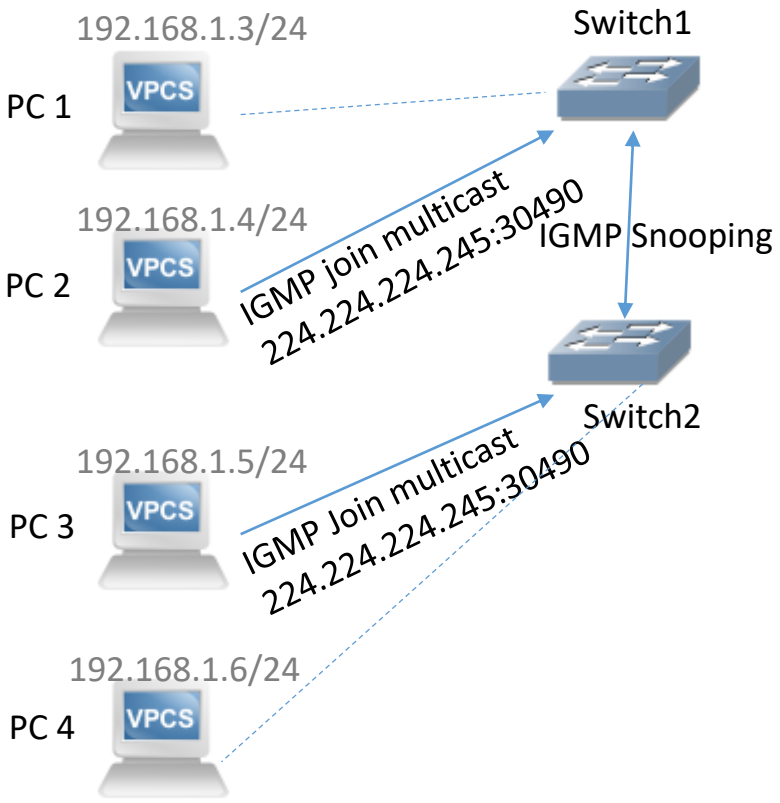
Bandwidth usage doubles for each additional receiver.

The **sender's CPU** must process multiple send operations.

Mutilcast is better for this case

2. SOME/IP run over Ethernet

➢ Mutilcast (UDP only)



Switch 1 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245: 30490	Port_PC2, Port_trunk_to_Switch2

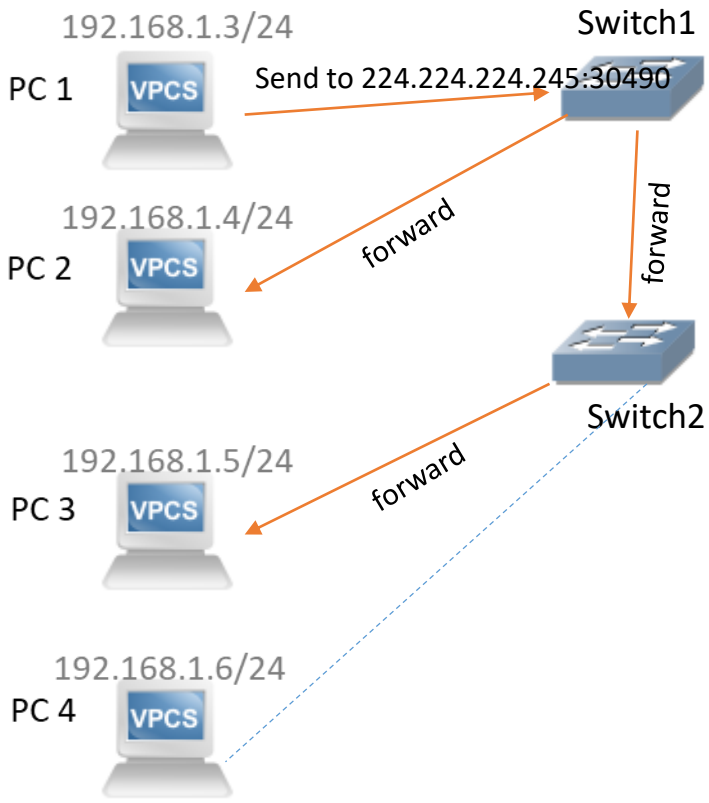
Switch 2 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245: 30490	Port_PC3, Port_trunk_to_Switch1

PC2 and PC3 join a multicast group (224.224.224.245:34090),
they will receive any data sent to that multicast IP address
— as long as the network infrastructure supports multicast forwarding.

2. SOME/IP run over Ethernet

➢ Mutilcast (UDP only)



Switch 1 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245: 30490	Port_PC2, Port_trunk_to_Switch2

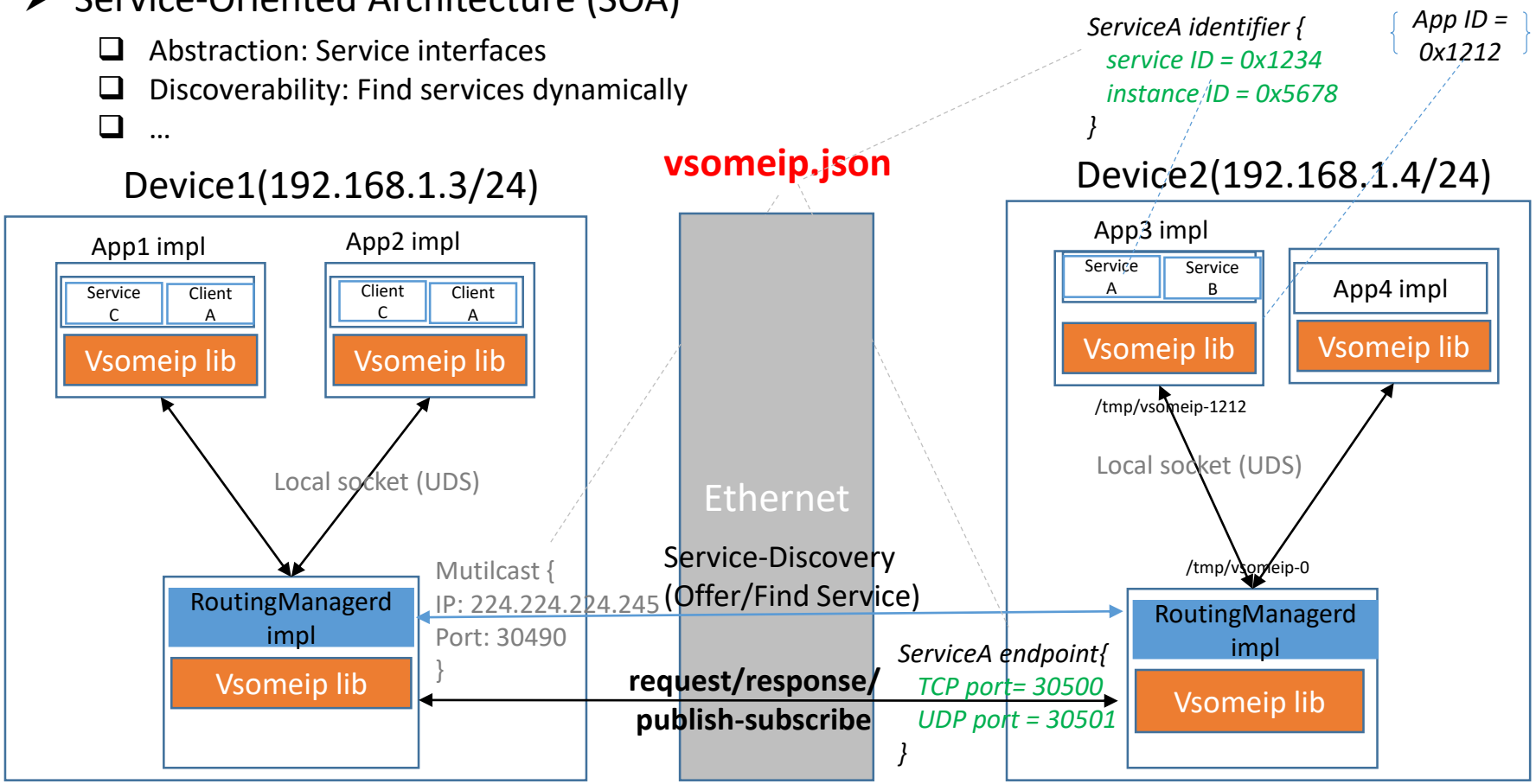
Switch 2 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245: 30490	Port_PC3, Port_trunk_to_Switch1

When PC1 sends data to a multicast IP address (224.224.224.245:30490),
and PC2 and PC3 have joined that multicast group, they will receive the same data.

3. Vsomeip

- Service-Oriented Architecture (SOA)
  - Abstraction: Service interfaces
  - Discoverability: Find services dynamically
  - …



ClientA don't know Device2 (IP + TCP/UDP Port)
ClientA only know ServiceA identifier (service ID + instance ID).
Q: How client A can send data to Service A ?

- The **RoutingManagerd** (Middleware daemon) is responsible for:
  Routing service messages between applications on the same device.
  Forwarding messages between devices over Ethernet.
  Managing **service discovery** and subscription handling.
  There is only one routingmanagerd per device/host.
- **vsomeip.json** define multiple critical purposes in the vsomeip architecture (include *ServiceA identifier, ServiceA endpoint*)

3. Vsomeip

- **Vsomeip.json** is the main configuration file used by the vsomeip runtime to define:
  https://github.com/COVESA/vsomeip/blob/master/documentation/vsomeipConfiguration.md

```
{
  "unicast" : “192.168.1.4",
  “logging” : {"level" : "debug", "console" : "true","dlt" : "false"}
  "applications" :
  [
    {
      "name" : "app3",
      "id" : "0x1212"
    }
  ],
  "services" :
  [
    {
      "service" : "0x1234", "instance" : "0x5678",
      "reliable": "30500",  "unreliable": "30501“,
      "events" : [ { "event" : "0x8778", "is_reliable" : "false", ... }],
      "eventgroups" :
      [
        {
                "eventgroup" : "0x4465",
                "multicast" : {"address" : "224.244.224.246", "port" : "30506" },
                "events" : [ "0x8778" ],
                "threshold" : "1"
        }
      ]
    }
  ],
  "routing" : "routingmanagerd",
  "service-discovery" :
  {
    "multicast" : "224.224.224.245",
    "port" : "30490",
    “ttl”: “3”
    "cyclic_offer_delay" : "2000",
    ...
```

1. Application Identity
Application name
Instance ID
Routing configuration (local or remote)

2. Service Definitions
Services offered (offer_service)
Services consumed (request_service)
Event groups and events

3. Transport Settings
TCP or UDP
Reliable vs unreliable communication
Port numbers and IP addresses

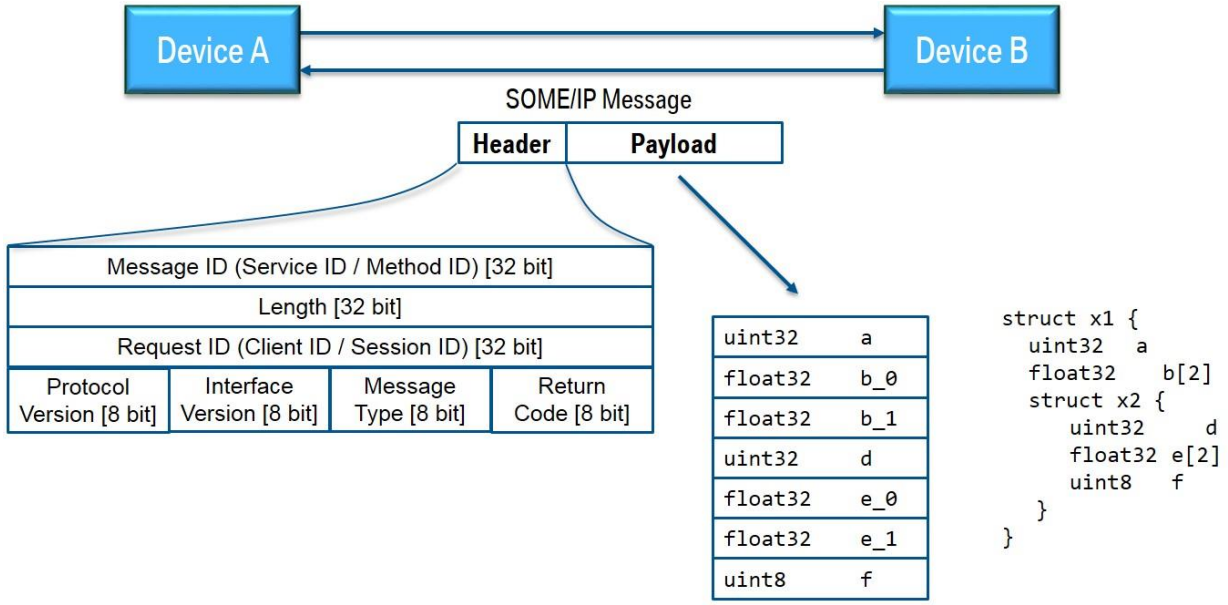
4. Service Discovery (SD)
Enable/disable SD
Repetition intervals for OfferService
TTL and re-subscription behavior
Multicast settings

5. Logging and Debugging
Log level
Console/file output
Tracing options

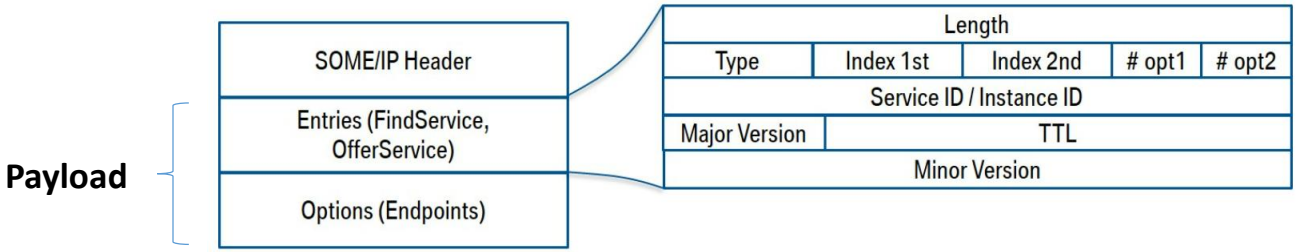
3. Vsomeip

➢ SOME/IP Message Format

- Header: includes information of who is requesting and which service is requested, and what is request
- Payload: contains the serialized data.



❑ SOME/IP Service-Discovery Message (Offer/ Find / Subscribe …)



3. Vsomeip

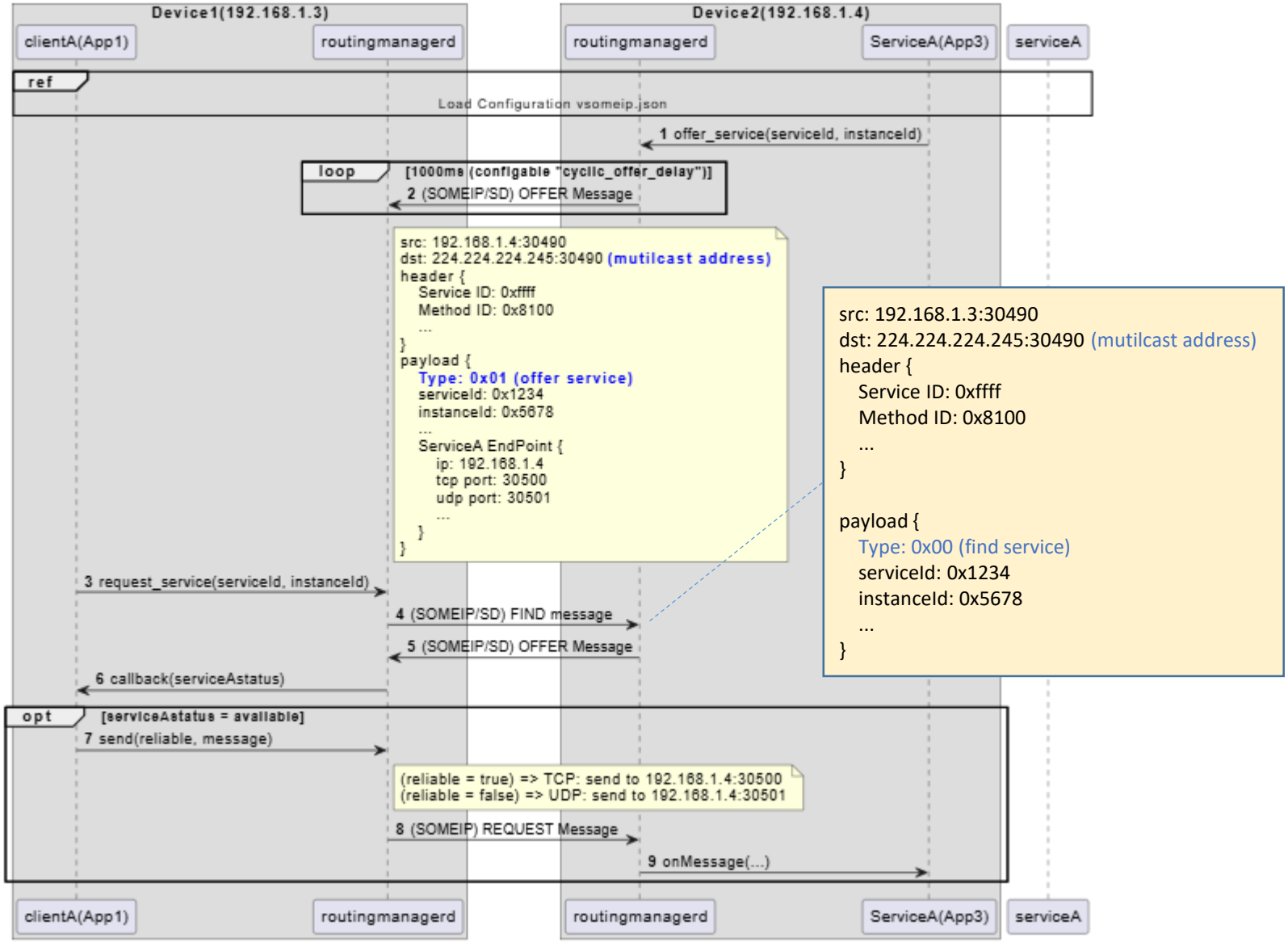
- SOME/IP header

https://github.com/COVESA/vsomeip/wiki/vsomeip-in-10-minutes

Field	Description
Service ID	unique identifier for each service
Method ID	0-32767 for methods, 32768-65535 for events
Length	length of payload in byte (covers also the next IDs, that means 8 additional bytes)
Client ID (App ID)	unique identifier for the calling client inside the ECU; has to be unique in the overall vehicle
Session ID	identifier for session handling; has to be incremented for each call
Protocol Version	0x01
Interface Version	major version of the service interface
Message Type	REQUEST (0x00) / REQUEST_NO_RETURN (0x01) / NOTIFICATION (0x02) / RESPONSE (0x80)/ REQUEST_ACK (0x40) / NOTIFICATION_ACK (0x42 / ERROR (0x81) / RESPONSE_ACK (0xC0 / RESPONSE_ACK (0xC0) / ERROR_ACK (0xC1) / UNKNOWN (0xFF)
Return Code	E_OK (0x00) / E_NOT_OK (0x01) / E_WRONG_INTERFACE_VERSION (0x08) / E_MALFORMED_MESSAGE (0x09) E_WRONG_MESSAGE_TYPE (0x0A) / E_UNKNOWN_SERVICE (0x02) / E_UNKNOWN_METHOD (0x03) / E_UNKNOWN_METHOD (0x03) E_NOT_READY (0x04) / E_NOT_REACHABLE (0x05) / E_NOT_REACHABLE (0x05) / E_TIMEOUT (0x06) E_WRONG_PROTOCOL_VERSION (0x07) / E_UNKNOWN (0xFF)

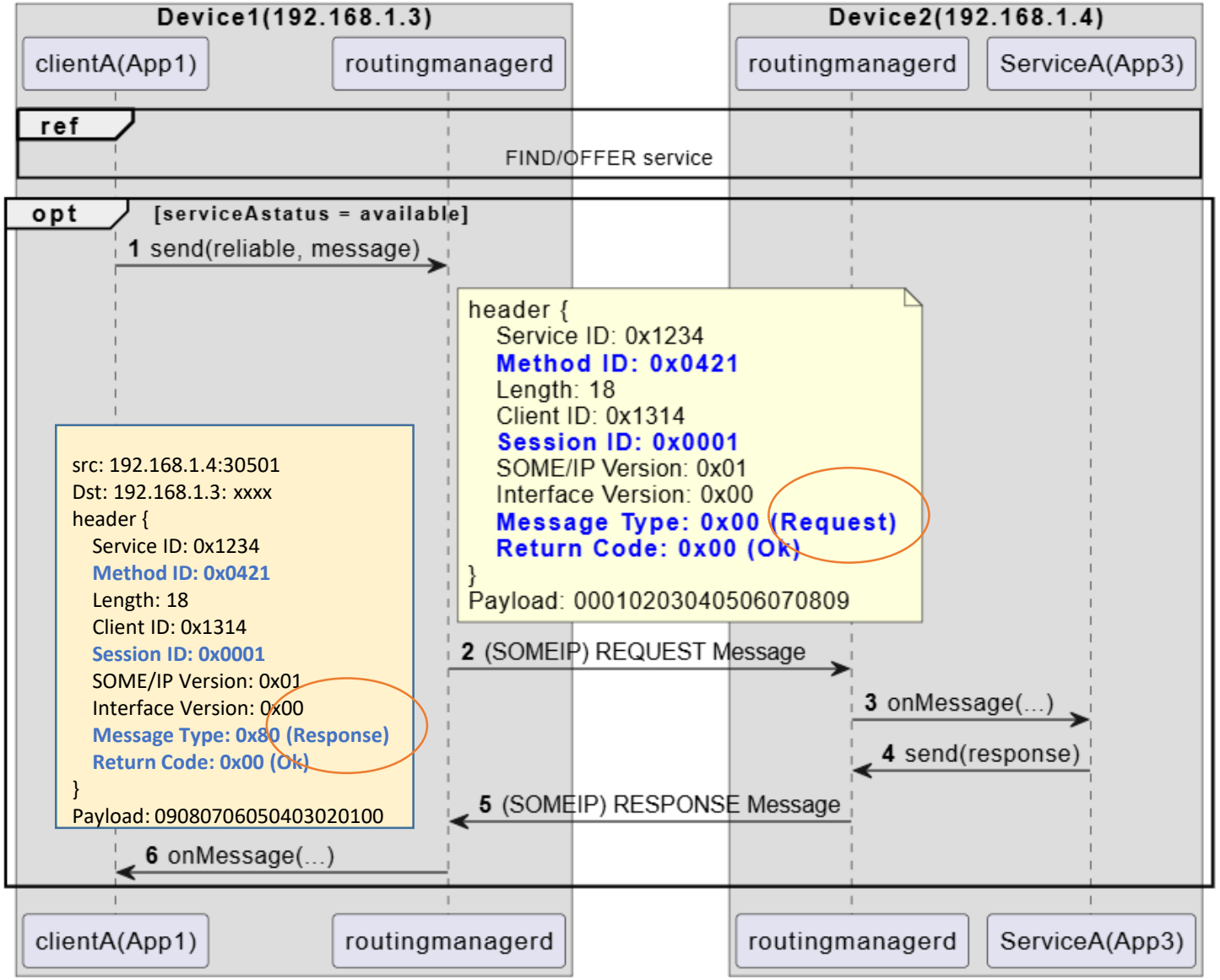
3. Vsomeip

- Service-Discovery (Offer/Find Service)



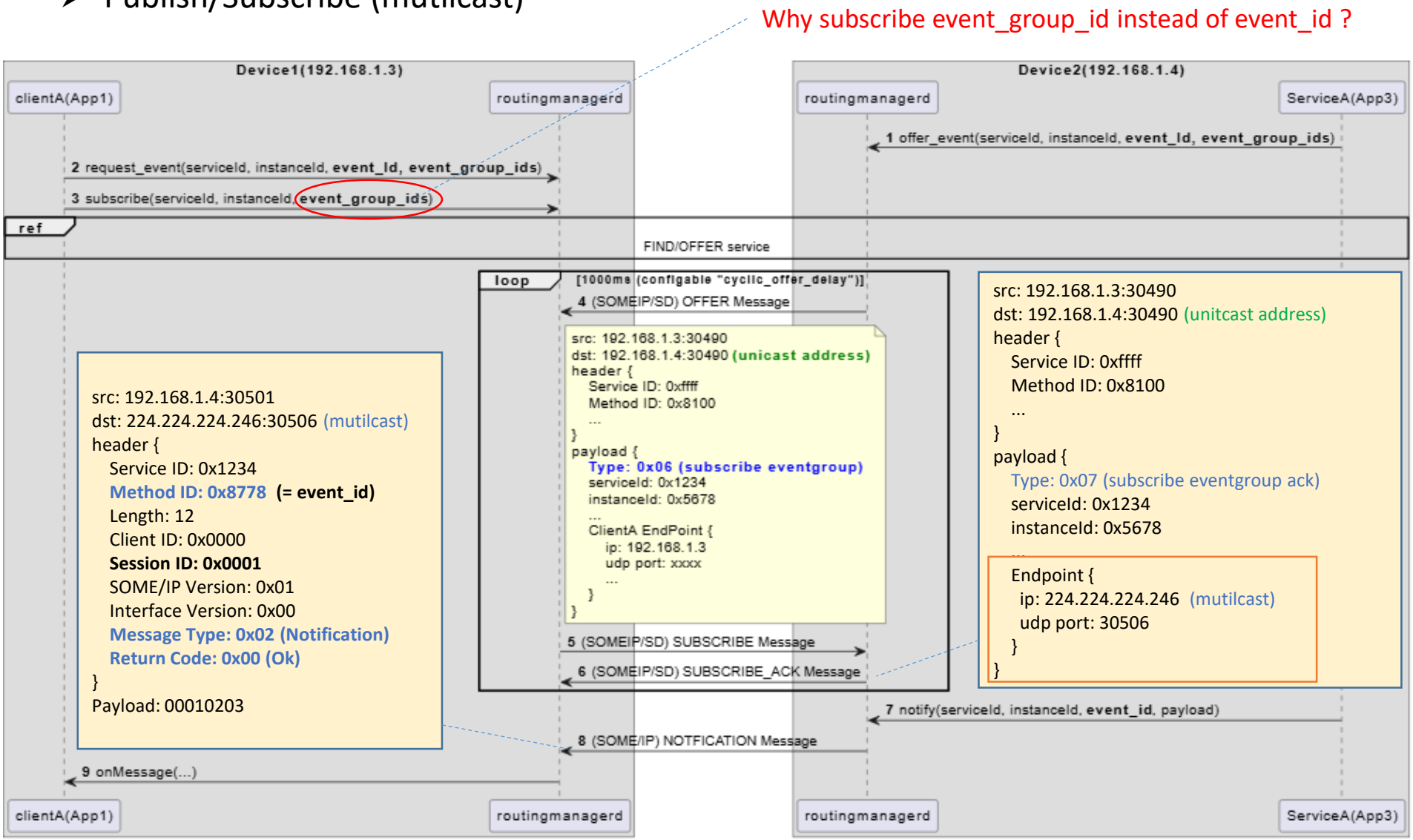
3. Vsomeip

- Request/Response (unicast)



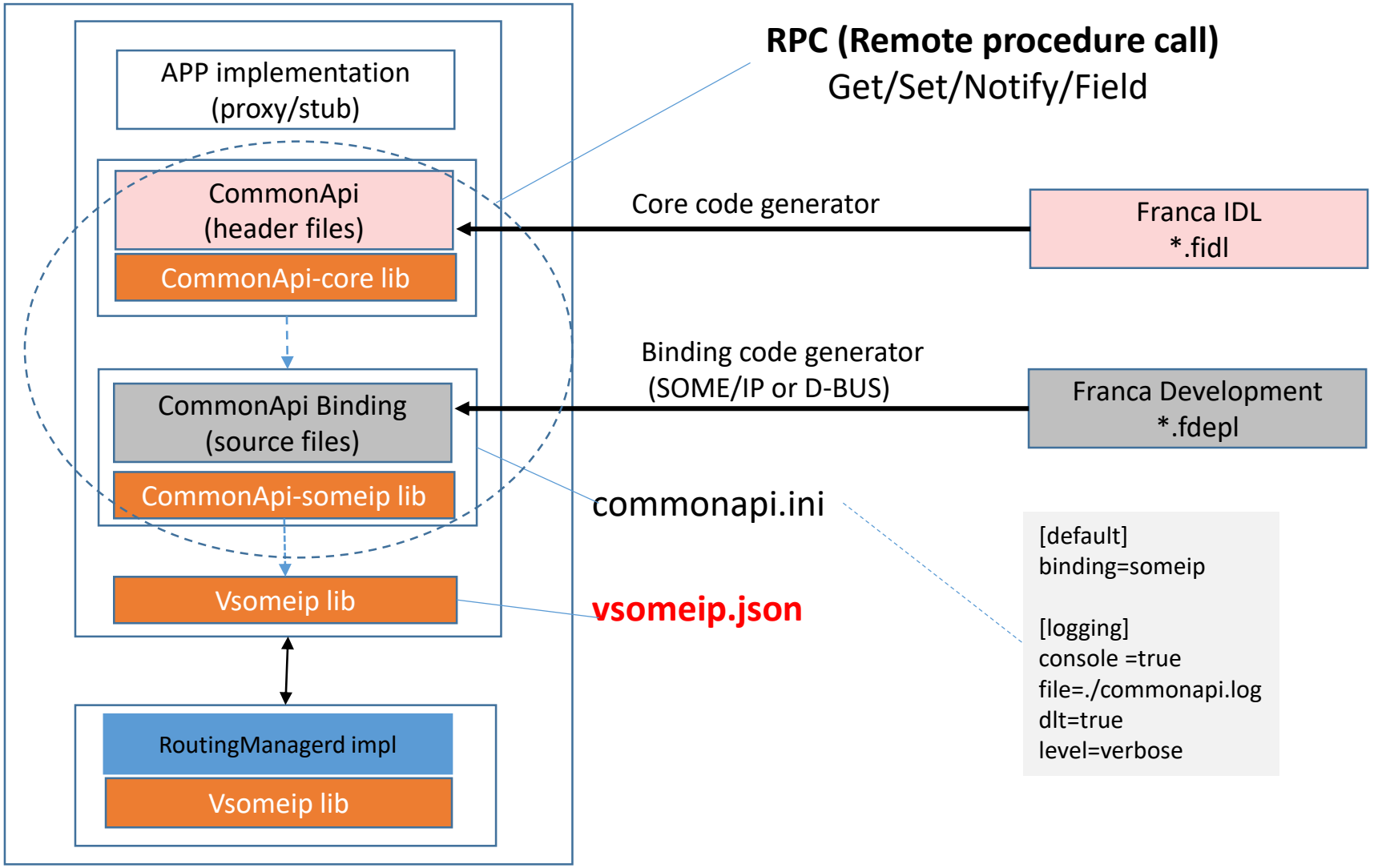
3. Vsomeip

- Publish/Subscribe (mutilcast)



4. Vsomeip + CommonApi

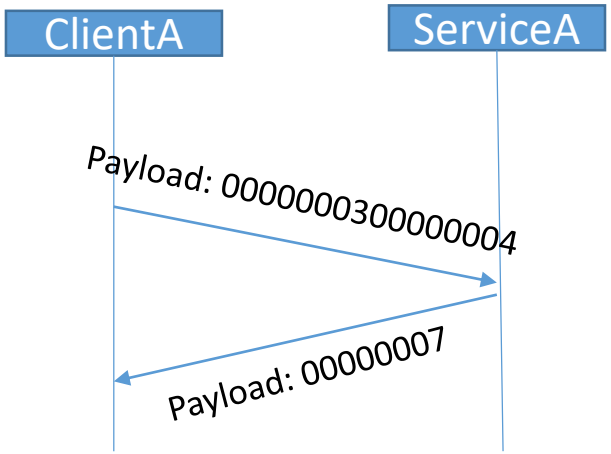
➢ **CommonAPI: A middleware abstraction layer that allows you to define services using Franca IDL and generate client/server code automatically**



4. Vsomeip + CommonApi

- **RPC (Remote procedure call)**
  - **GET Method**

    Int32_t calculateSum(int32_t a, int32_t b);



Request(0x00) / response(0x80)

```
…
Int32_t a = 3;
Int32_t b = 4;
Int32_t sum = 0;
CommonAPI::CallInfo info(3000ms); //timeout default = 5s
myProxy->calculateSum(a, b, callStatus, sum , &info);
Printf("sum = %d", sum ); // sum = 7
// async call
myProxy->calculateSumAsync(a, b, callback, &info);
```

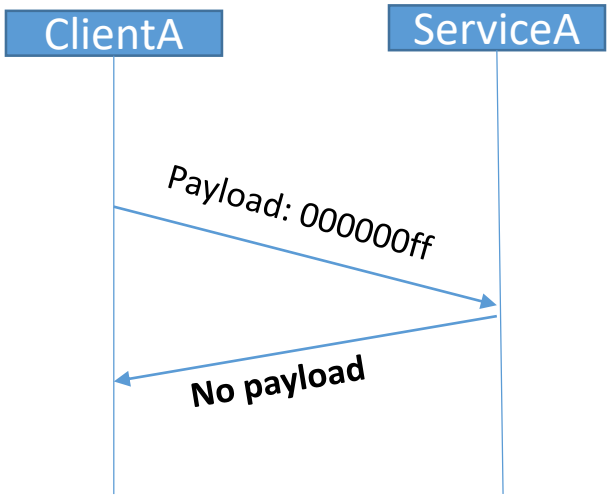
```
…
XXXStubImpl::calculateSum(
   const std::shared_ptr<CommonAPI::ClientId> _client,
   int32_t a, int32_t b, calculateSumReply_t _reply) {

   int32_t sum = a + b;
   _reply(sum );
}
```

4. Vsomeip + CommonApi

- **RPC (Remote procedure call)**
  - **SET Method**
    void setValue(int32_t x);

CommonApi-someip does not support fire&forget
⇒ Client send message with Type = 0x00(REQUEST) instead of Type = 0x01(REQUEST_NO_RETURN)



Request(0x00) / response(0x80)

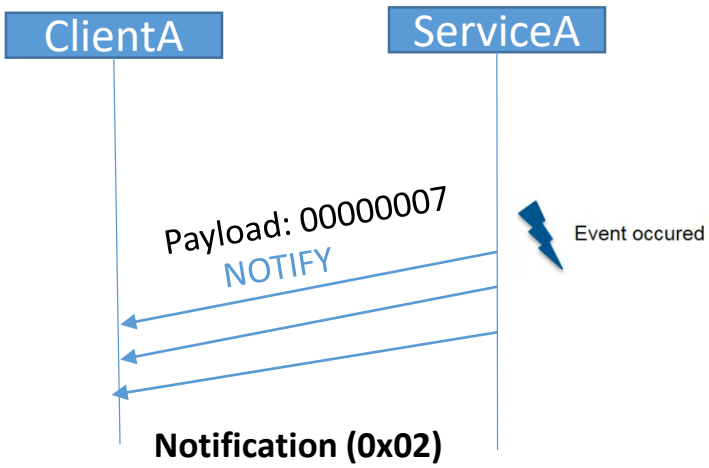
```
…
Int32_t x = 255;
myProxy->setValue( x, callStatus, &info);
```

```
…
XXXStubImpl::setValue(
   const std::shared_ptr<CommonAPI::ClientId> _client,
   int32_t x, setValueReply_t _reply) {

   printf("x = %d\n", x);
   _reply( ); // if comment this line => client get callstatus = REMOTE_ERROR
}
```

5. Vsomeip + CommonApi

- **RPC (Remote procedure call)**
  - **NOTIFY Method (pub-sub)**
    void onMyStatusChanged (int32_t myStatus);




```
…
// subscribe
myProxy->getMyStatusEvent().subscribe(onMyStatusChanged);
```

```
…
Int myStatus = 7;
// notify
myService->fireMyStatusEvent(myStatus );
```

Event – a Fire&Forget callback, that is sent out by the Server (e.g. cyclically or on change).

❖ In SOME/IP, if Service Discovery (SD) is disabled, then notification (event communication) is not supported in the dynamic subscription model.

5. Vsomeip + CommonApi

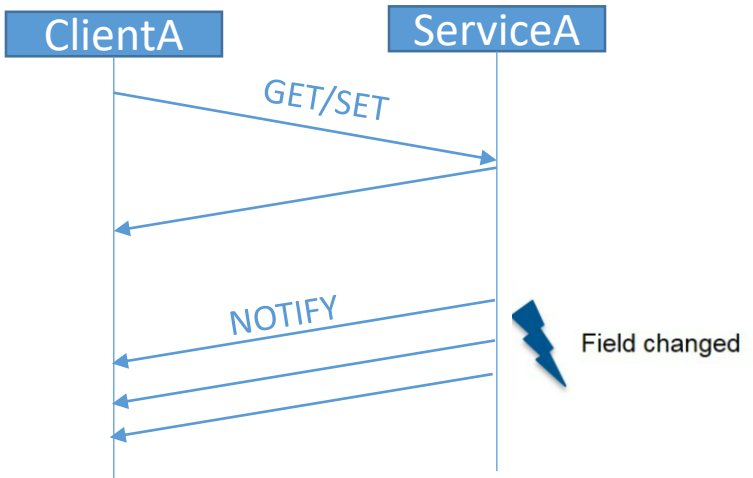
➢ RPC (Remote procedure call)

❑ **Field** – represents a remote accessible property that includes Getter/Setter and/or Notification.

- **Getter** – Method to read field value.
- **Setter** – Method to set field value.
- **Notification** (sends out Events with new values on change of field value, notification is also sent immediately when a client first registers for the attribute (initial value)).

StudentData getStudentData()
void setStudentData(StudentData v)
void onStudentDataChanged(StudentData v)

```
…
StudentData data;
//getter
myProxy->getStudentDataAttribute().getValue(callStatus, data, &info);
//setter
myProxy->getStudentDataAttribute().setValue(data, callStatus, data, &info);
//Subscribe
 myProxy->getStudentDataAttribute().getChangedEvent().
subscribe(onStudentDataChanged)
```



Request(0x00) / response(0x80) / notification(0x02)

```
…
StudentStruct  data = xxx;
//notify
myService->setStudentDataAttribute(data);
```

❖ In SOME/IP, if Service Discovery (SD) is disabled, then notification (event communication) is not supported in the dynamic subscription model.

6. Demo

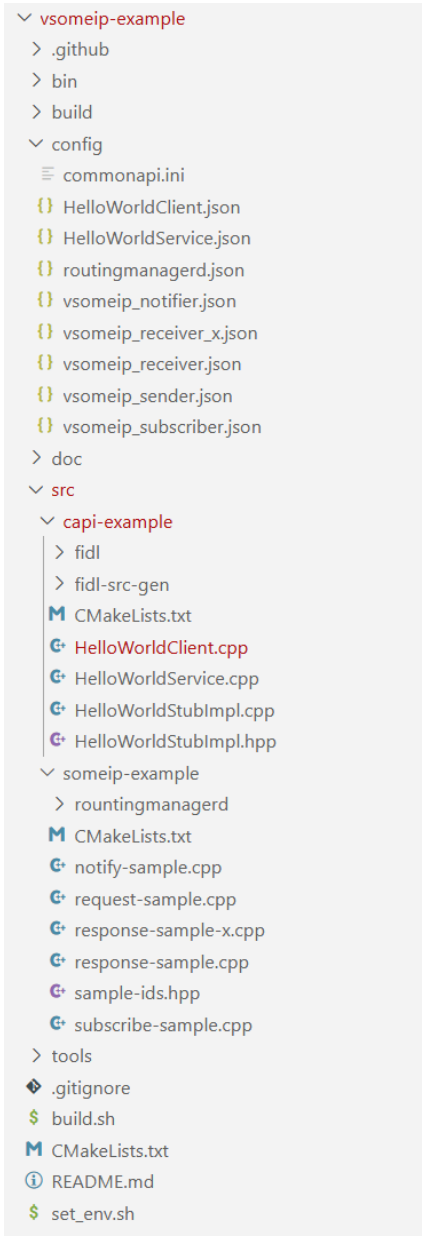
➢ Download & Build example

Environment: Ubuntu 18.04 and higher

1) Download vsomeip-example
 # Install essential tools
sudo apt update
sudo apt install -y openssh-server screen git
sudo apt install -y net-tools netcat socat tcpdump

 # Clone example and install dependencies
git clone https://github.com/minhthedt/vsomeip-example.git
cd vsomeip-example
./set_env.sh

2) Build vsomeip-example
cd vsomeip-example
./build.sh
Output: ./vsomeip-example/bin/...



6. Demo

- [Important] Make sure both PCs can send/receive multicast before testing.

 # (Optional) Route all multicast traffic (224.0.0.0/4) via eth0
ifconfig
sudo ip route add 224.0.0.0/4 dev eth0
# or: sudo route add -net 224.0.0.0/4 dev eth0

 # Show routing table
netstat -rn

```
worker@00c0da233959:~/TRAINING/vsomeip-example$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG        0 0          0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U         0 0          0 eth0
224.0.0.0       0.0.0.0         240.0.0.0       U         0 0          0 eth0
worker@00c0da233959:~/TRAINING/vsomeip-example$
```

```
# Sender
echo "Hello multicast" | socat - UDP4-DATAGRAM:239.0.0.1:12345

# Receiver
socat -v UDP4-RECVFROM:12345,ip-add-membership=239.0.0.1:0.0.0.0,fork -
```

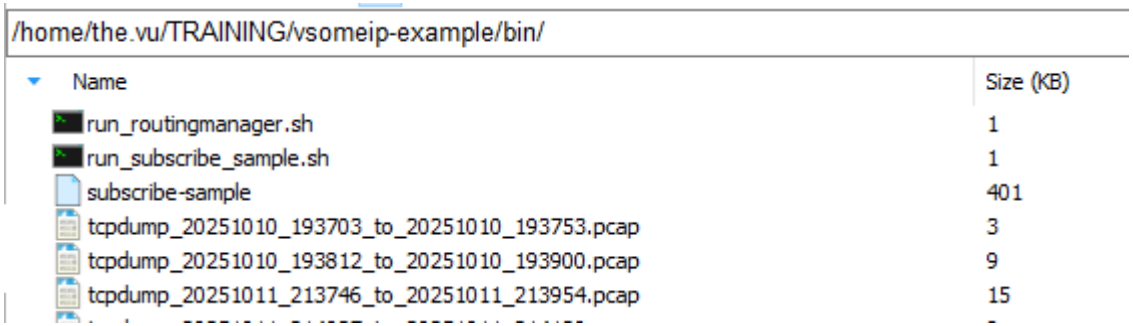
```
worker@00c0da233959:~/TRAINING/vsomeip-example$ socat -v UDP4-RECVFROM:12345,ip-add-membership=239.0.0.1:0.0.0.0,fork -
> 2025/10/13 18:24:01.808449  length=16 from=0 to=15
Hello multicast
Hello multicast
```

6. Demo

➢ Run Tests

PC1 (172.17.0.6)	**PC2**(172.17.0.5)
Set correct IP addresses to ./*vsomeip-example*/config/HelloWorldClient.json	**Set correct IP addresses to** ./*vsomeip-example*/config/HelloWorldService.json
cd vsomeip-example *sudo ./bin/run_HelloWorldClient.sh*	*cd vsomeip-example* *sudo ./bin/run_HelloWorldService.sh*

❖ Pcap log will be generated after press “ctrl + C” to terminate program.

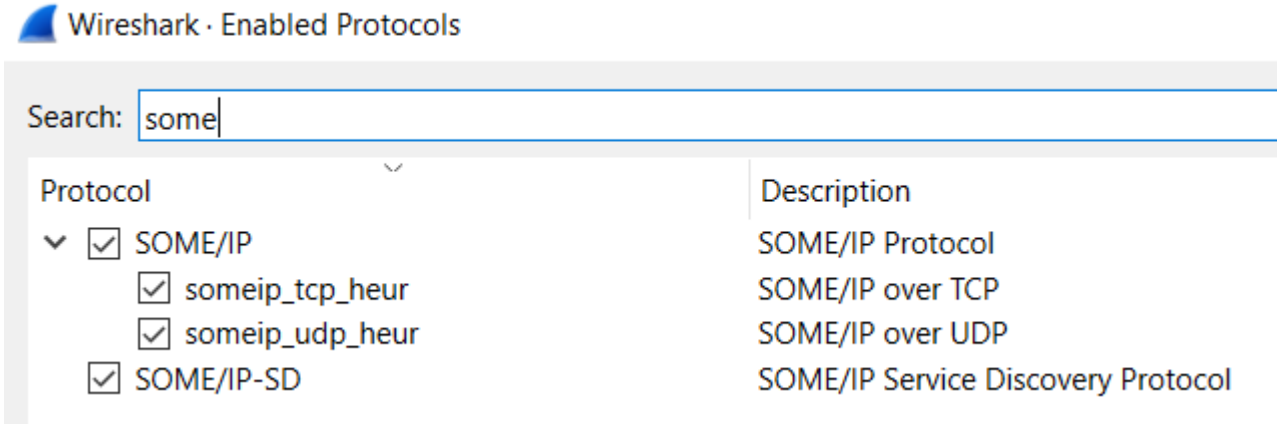


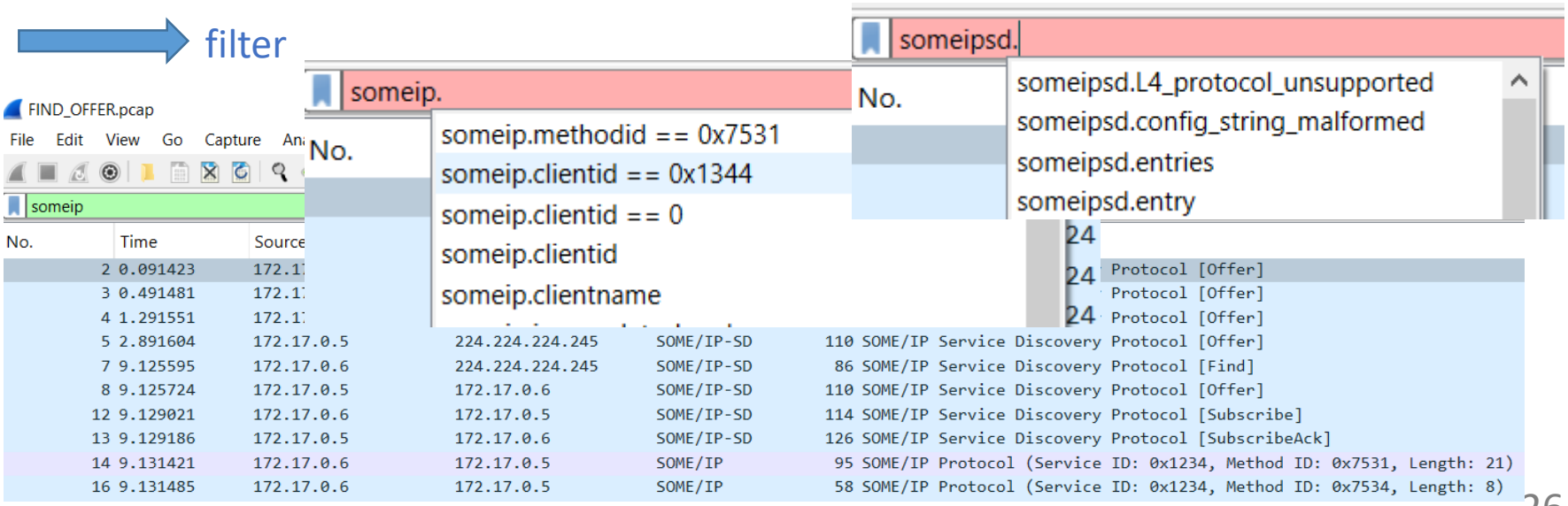
Example log:
vsomeip-example/doc/FIND_OFFER.pcap
vsomeip-example/doc/REQUEST_RESPONSE.pcap
vsomeip-example/doc/ HelloWorldService.pcap
vsomeip-example/doc/ HelloWorldService_DLT_LOG.zip

6. Demo

➢ Analyze pcap log

- Install writeshark (https://www.wireshark.org/download.html)
- Open wireshark -> Analyze -> Enable protocols … -> tick box SOME/IP, SOME/IP-SD





6. Demo

➢ Analyze DLT log

ECU1 E01C TC 1631380 **nw_trace** ipc verbose 2 ac 11 00 05 77 33 02 01 56 78 12 34 75 32 00 00 00 10 13 13 00 02 01 00 00 00 00 00 00 03 00 00 00 04
ECU1 E01C TC 1631380 nw_trace ipc verbose 2 ac 11 00 05 77 33 02 00 56 78 12 34 75 32 00 00 00 0c 13 13 00 02 01 00 80 00 00 00 00 07

Byte(s)	Value	Meaning
ac 11 00 05	172.17.0.5	IPv4 address (Docker container IP)
77 33	30515	Ephemeral client port
02	UDP	Protocol (0x02 = UDP)
01	Sending	Direction (vsomeip is sending the message)
56 78	0x5678	Instance ID (HelloWorld instance)

Byte(s)	Value	Meaning
12 34	0x1234	Service ID (HelloWorld service)
75 32	0x7532	Method/Event ID
00 00 00 10	16 bytes	Length of header remainder + payload
13 13	0x1313	Client ID
00 02	0x0002	Session ID
01	1	Protocol Version (SOME/IP v1.0)
00	0	Interface Version
00	REQUEST	Message Type (0x00 = REQUEST)
00	E_OK	Return Code
00 00 00 **00 00 00 03** **00 00 00 04**	Payload	11 bytes of data

HelloWorldClient.json

```
"tracing" :
{
   "enable" : "true"
},
```

tracing of the SOME/IP messages is enabled
-> log the internal messages that are sent over the Unix Domain Sockets (UDS) to DLT

7. Q&A

Question	Answer
Why routingmanagerd need to send “OFFER” periodically ? "service-discovery" : { "ttl" : "3", ...}	Each OFFER includes a TTL(Time To Live) value. If clients don’t receive a new OFFER before TTL expires, they assume the service is **unavailable**. Periodic offers **refresh the TTL**, keeping the service marked as active.
Why routingmanagerd need to send “SUBSCRIBE” periodically ?	Each subscription has a **Time-To-Live (TTL)**. If the server doesn’t receive a renewed subscription before TTL expires, it **removes the client** from the event group. Periodic SUBSCRIBE messages **refresh the TTL**, keeping the subscription alive.
If disable service-discovery, what happen ?	“notify event” is not supported Request/respon work normally if client know specific IP/port of service "services" : [{ "service" : "0x1234", "instance" : "0x5678", "unicast" : "172.17.0.5", "unreliable" : "30509",...}]
Can service notify events via both unicast/mutilcast ?	YES (demo in HelloWorldService.json)
How’s about SOME/IP security ?	https://github.com/COVESA/vsomeip/blob/master/documentation/vsomeipConfiguration.md#security

8. Reference

https://github.com/COVESA/vsomeip
https://github.com/COVESA/vsomeip/wiki/vsomeip-in-10-minutes#devices
https://github.com/COVESA/vsomeip/tree/master/documentation

https://github.com/COVESA/capicxx-someip-tools/wiki/CommonAPI-C---SomeIP-in-10-minutes
https://github.com/GENIVI/capicxx-core-tools/releases/download/3.1.12/CommonAPICppUserGuide.pdf
https://github.com/COVESA/capicxx-someip-tools/blob/master/org.genivi.commonapi.someip/src-gen/org/genivi/commonapi/someip/Deplo