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 & Q&A**

   https://github.com/minhthedt/vsomeip-example

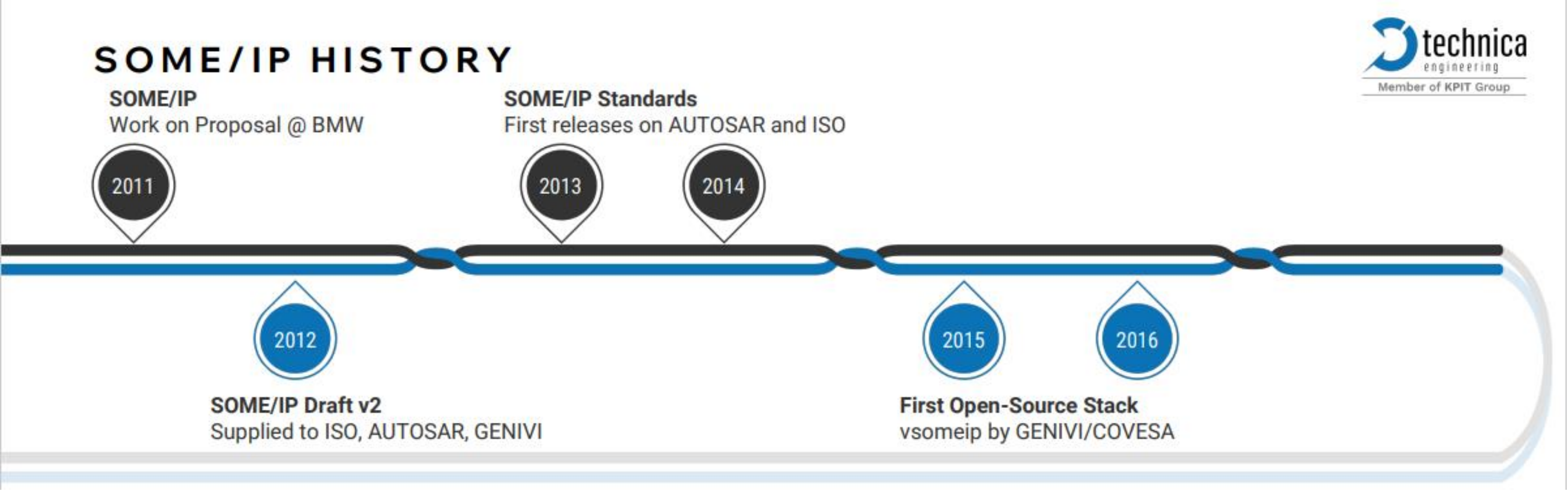
Presenter the.vu@lge.com, Ha Noi 2025-10-03

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**

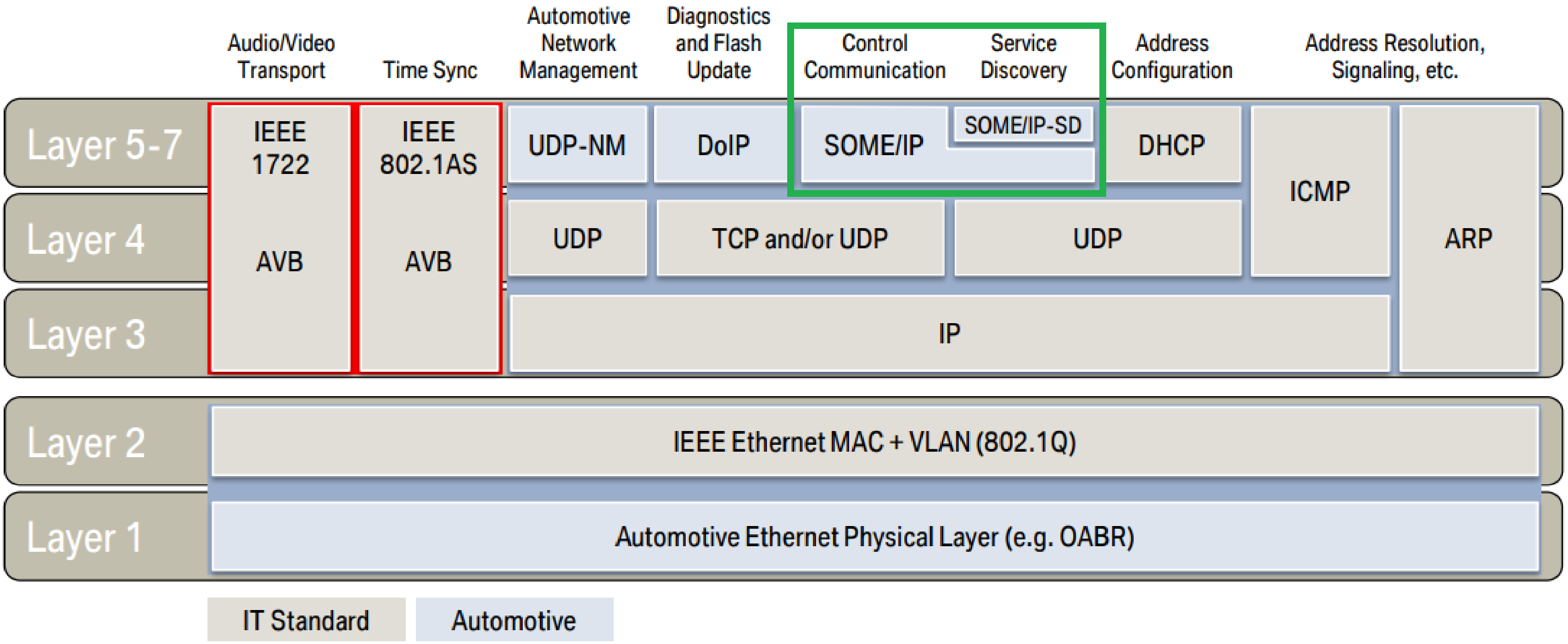
SOME/IP remained as a **proprietary protocol**, originally developed and published by **BMW Group** for internal use and later shared publicly.

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.

Dr. Lars Völker, BMW Group, Communication Protocols for Ethernet in the vehicle, 2013-12-09

Page 7

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**

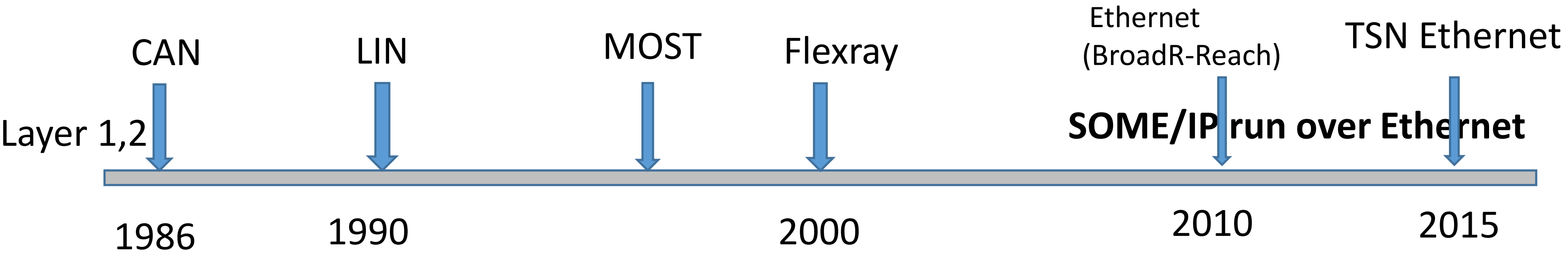
Engine Control Module (ECM)
Transmission Control Module (TCM)
Brake Control Module (ABS/ESC)
Airbag Control Unit
Body Control Module (BCM)
Steering Control Module …

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

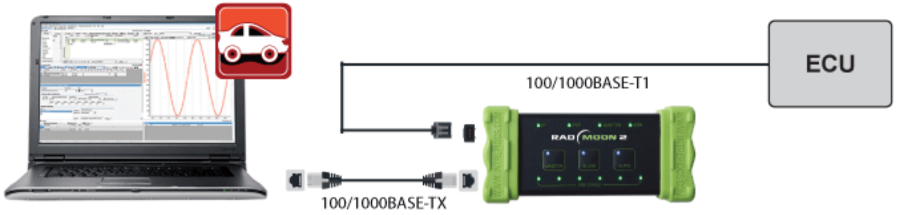
2. SOME/IP run over Ethernet

➢ 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

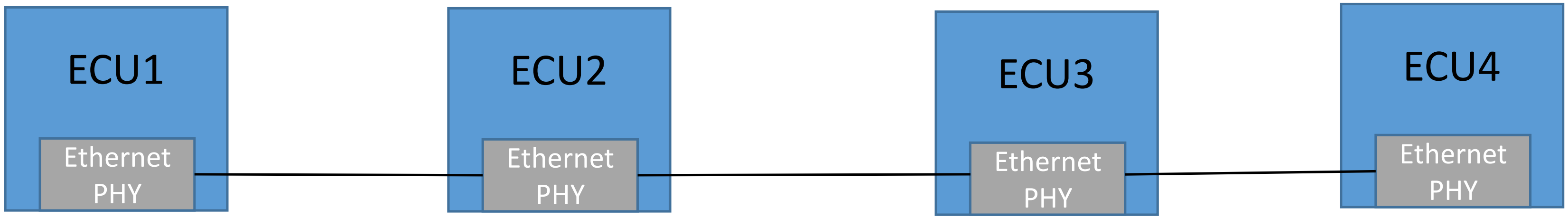
➢ Convert Automotive Ethernet <-> Traditional Ethernet: Media Converter



2. SOME/IP run over Ethernet

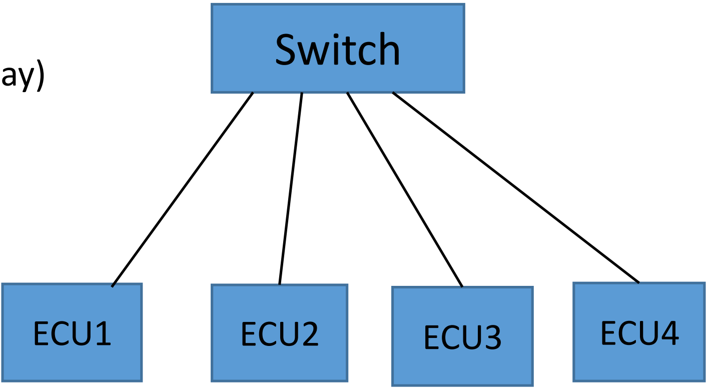
➢ Automotive Ethernet: Topology

❑ **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



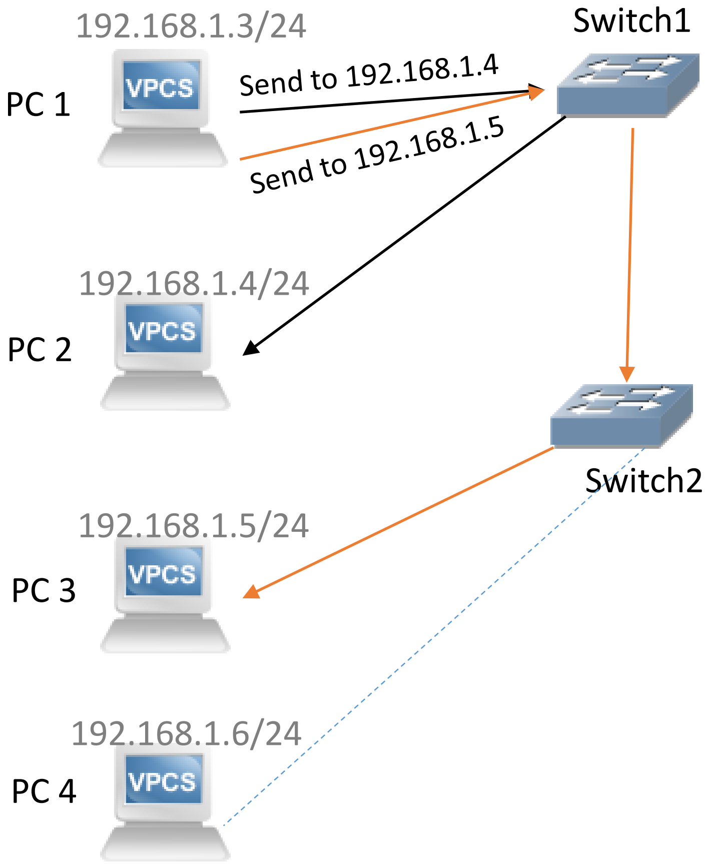
❑ **Star topology** (**most common today**)**:** used in most current automotive Ethernet architectures
Advantages:
Simple management
Easier diagnostics (one link failure doesn't affect others)
Supports mixed networks (Ethernet ↔ CAN, LIN, FlexRay via gateway)

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

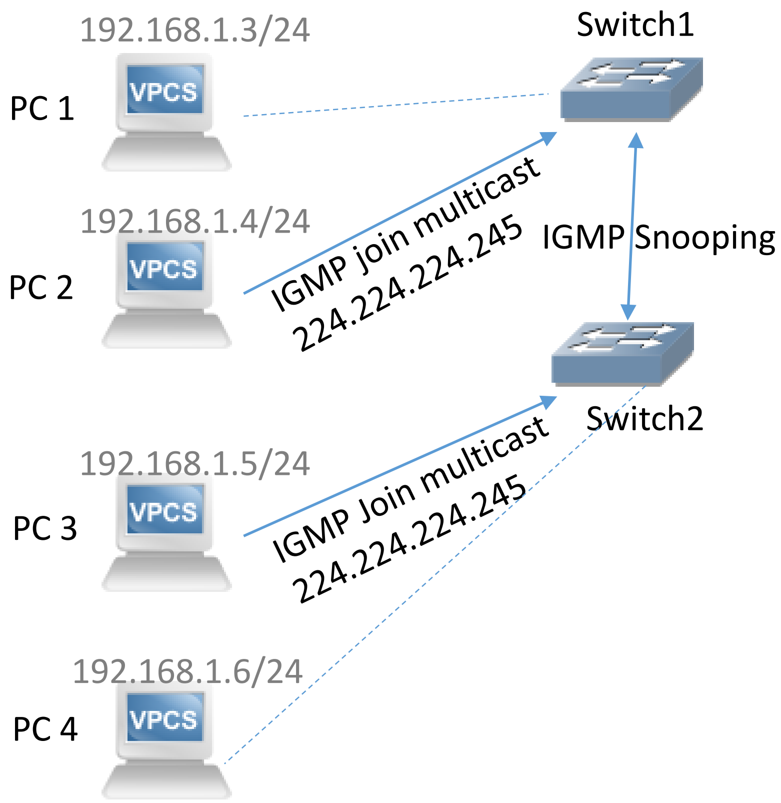
➡ This means:
2 copies of the same data are transmitted.
Bandwidth usage doubles for each additional receiver.
The **sender's CPU** must process multiple send operations.

⬇

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	Port_PC2, Port_trunk_to_Switch2

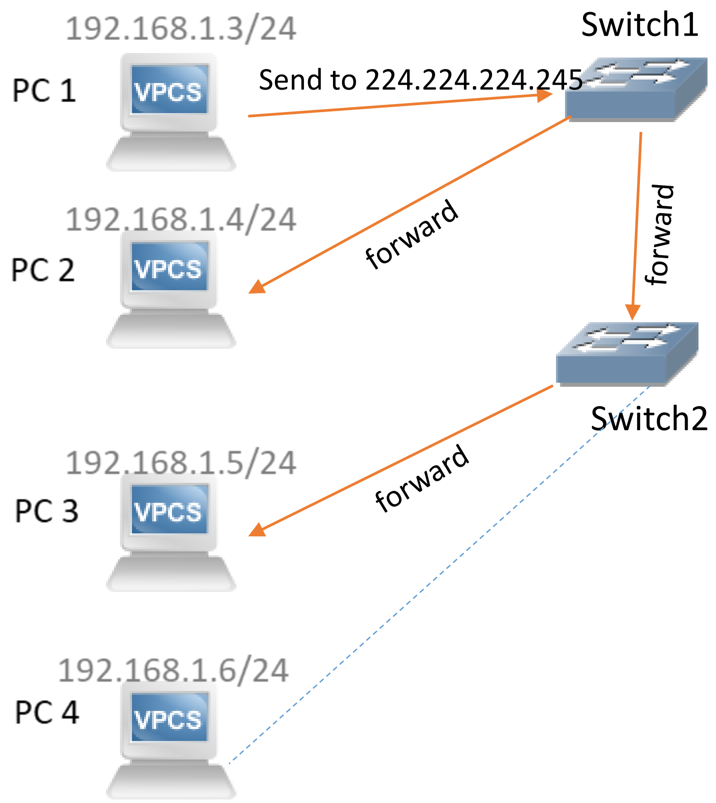
Switch 2 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245	Port_PC3, Port_trunk_to_Switch1

PC2 and PC3 join a multicast group (224.224.224.245),
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	Port_PC2, Port_trunk_to_Switch2

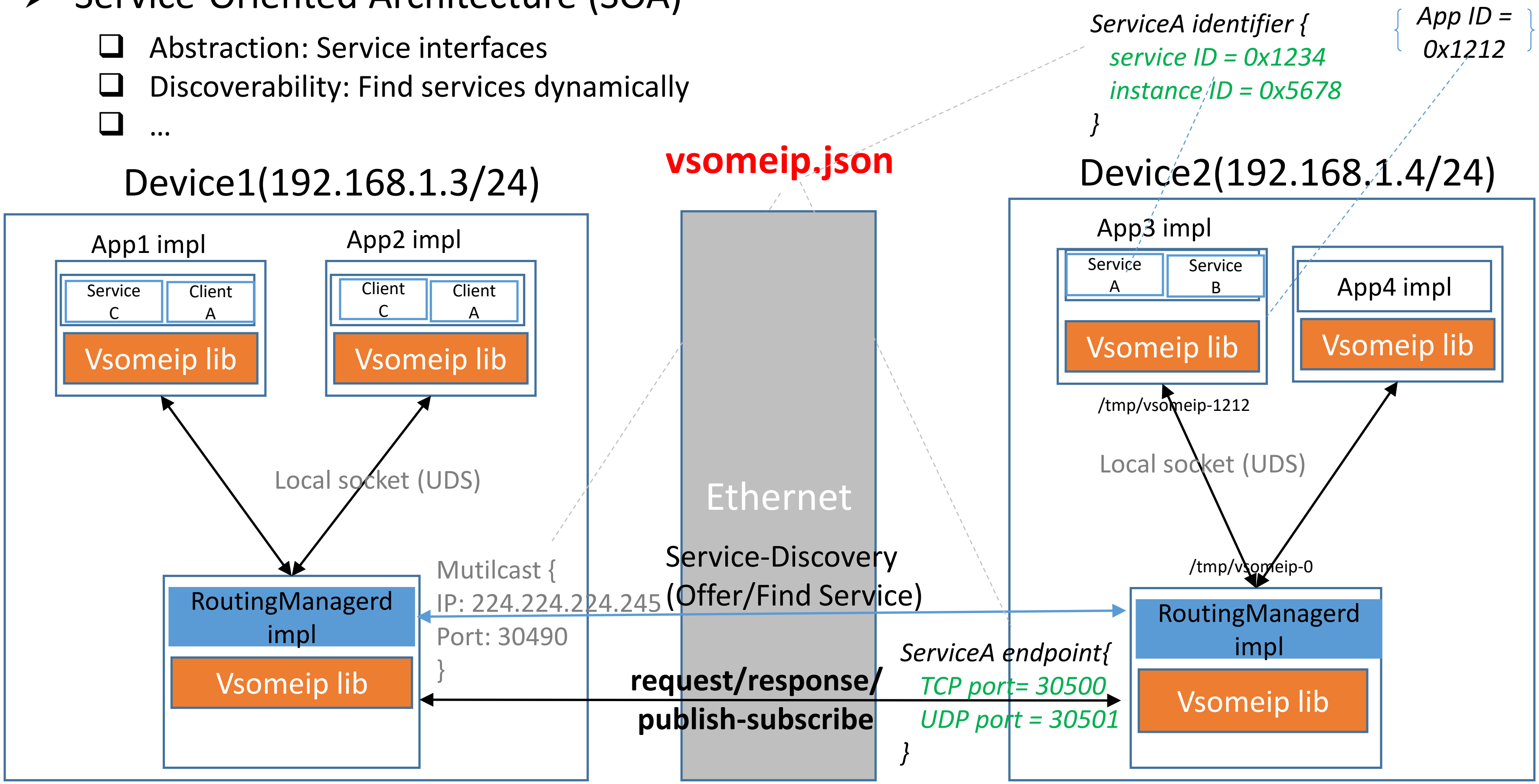
Switch 2 update multicast forwarding table

Multicast Group	Forward to ports
224.224.224.245	Port_PC3, Port_trunk_to_Switch1

When PC1 sends data to a multicast IP address (224.224.224.245),
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

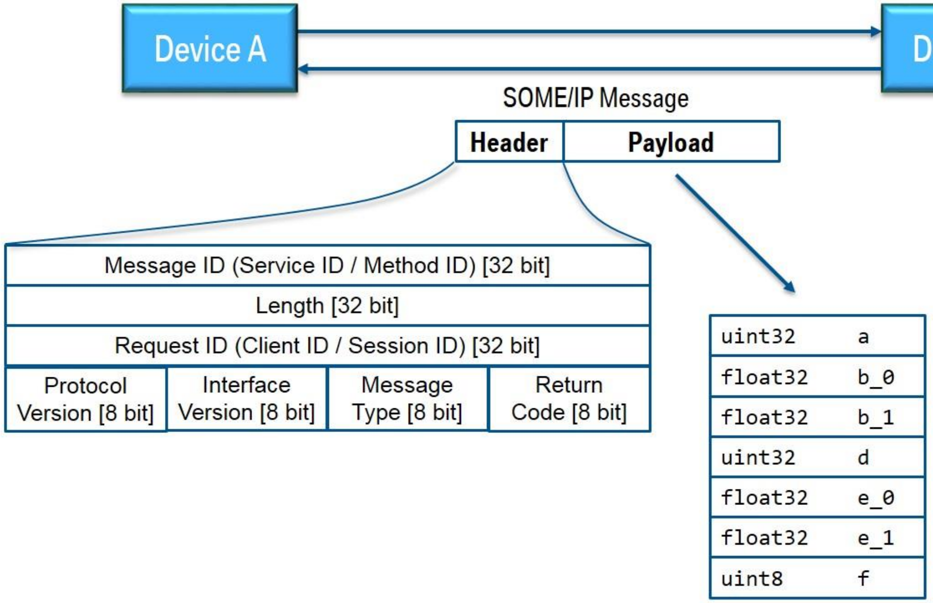
4. Service Discovery (SD)
Enable/disable SD
Repetition intervals for OfferService
TTL and re-subscription behavior
Multicast settings

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.

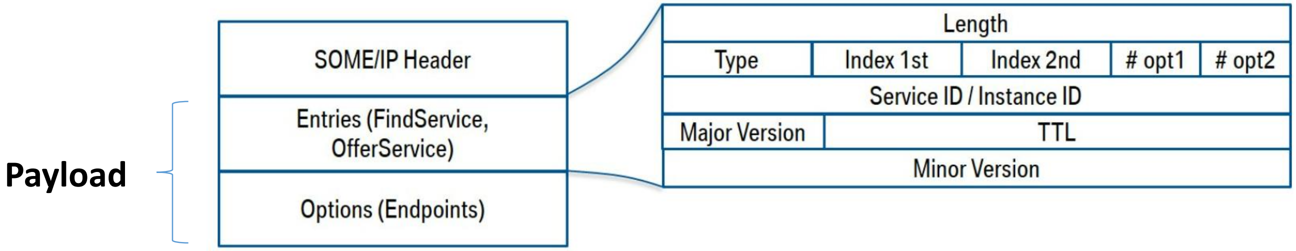


Message ID (Service ID / Method ID) [32 bit]			
Length [32 bit]			
Request ID (Client ID / Session ID) [32 bit]			
Protocol Version [8 bit]	Interface Version [8 bit]	Message Type [8 bit]	Return Code [8 bit]

uint32	a
float32	b_0
float32	b_1
uint32	d
float32	e_0
float32	e_1
uint8	f

```
struct x1 {
  uint32   a
  float32    b[2]
  struct x2 {
      uint32     d
      float32 e[2]
      uint8    f
   }
}
```

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



SOME/IP Header
Entries (FindService, OfferService)
Options (Endpoints)

Length				
Type	Index 1st	Index 2nd	# opt1	# opt2
Service ID / Instance ID				
Major Version	TTL			
Minor Version				

- IPv4 Address
- 0x06: TCP, 0x11: UDP)
- Port

3. Vsomeip

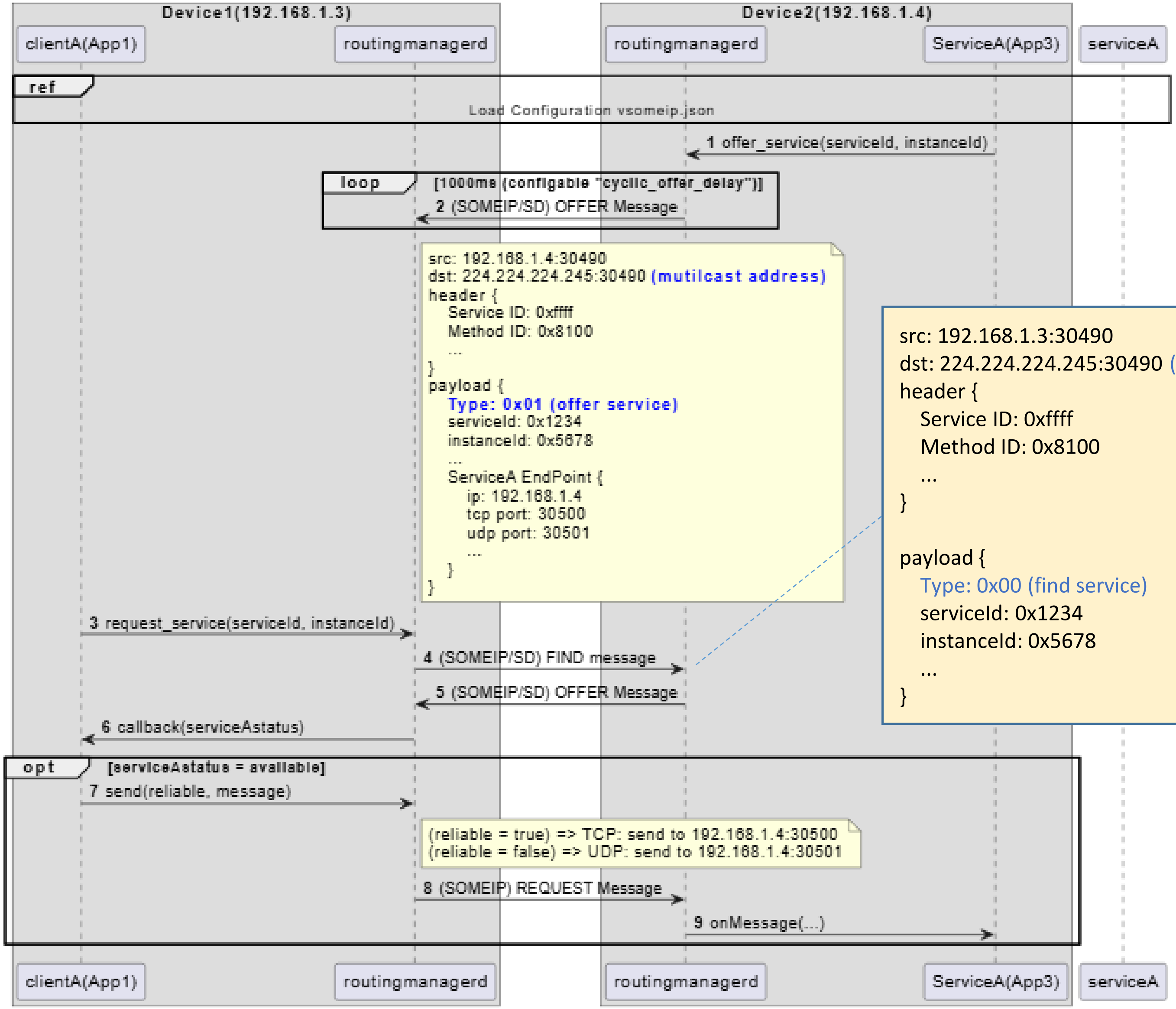
- SOME/IP header

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)


Device1(192.168.1.3)
Device2(192.168.1.4)
clientA(App1)
routingmanagerd
routingmanagerd
ServiceA(App3)
serviceA
ref
Load Configuration vsomeip.json
1 offer_service(serviceId, instanceId)
loop
[1000ms (configable "cyclic_offer_delay")]
2 (SOMEIP/SD) OFFER Message
3 request_service(serviceId, instanceId)
4 (SOMEIP/SD) FIND message
5 (SOMEIP/SD) OFFER Message
6 callback(serviceAstatus)
opt
[serviceAstatus = available]
7 send(reliable, message)
8 (SOMEIP) REQUEST Message
9 onMessage(...)
clientA(App1)
routingmanagerd
routingmanagerd
ServiceA(App3)
serviceA

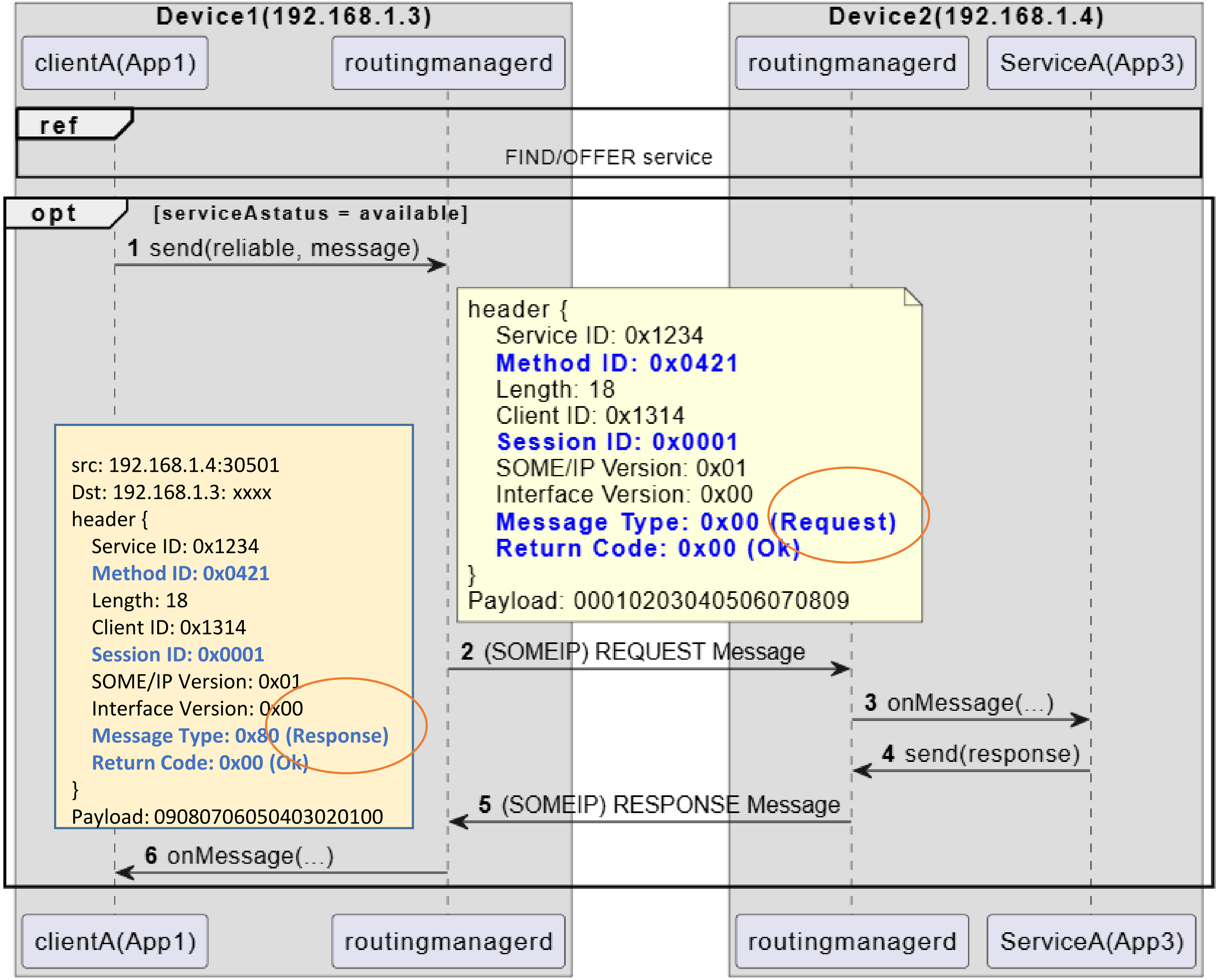
```
src: 192.168.1.4:30490
dst: 224.224.224.245:30490 (mutilcast address)
header {
   Service ID: 0xffff
   Method ID: 0x8100
   ...
}
payload {
   Type: 0x01 (offer service)
   serviceId: 0x1234
   instanceId: 0x5678
   ...
   ServiceA EndPoint {
      ip: 192.168.1.4
      tcp port: 30500
      udp port: 30501
      ...
   }
}
```

```
(reliable = true) => TCP: send to 192.168.1.4:30500
(reliable = false) => UDP: send to 192.168.1.4:30501
```

```
src: 192.168.1.3:30490
dst: 224.224.224.245:30490 (mutilcast address)
header {
   Service ID: 0xffff
   Method ID: 0x8100
   ...
}

payload {
   Type: 0x00 (find service)
   serviceId: 0x1234
   instanceId: 0x5678
   ...
}
```

3. Vsomeip

- Request/Response (unicast)



Device1(192.168.1.3)

clientA(App1)

routingmanagerd

Device2(192.168.1.4)

routingmanagerd

ServiceA(App3)

ref

FIND/OFFER service

opt [serviceAstatus = available]

1 send(reliable, message)

```
header {
    Service ID: 0x1234
    Method ID: 0x0421
    Length: 18
    Client ID: 0x1314
    Session ID: 0x0001
    SOME/IP Version: 0x01
    Interface Version: 0x00
    Message Type: 0x00 (Request)
    Return Code: 0x00 (Ok)
}
Payload: 00010203040506070809
```

2 (SOMEIP) REQUEST Message

3 onMessage(...)

4 send(response)

5 (SOMEIP) RESPONSE Message

```
src: 192.168.1.4:30501
Dst: 192.168.1.3: xxxx
header {
    Service ID: 0x1234
    Method ID: 0x0421
    Length: 18
    Client ID: 0x1314
    Session ID: 0x0001
    SOME/IP Version: 0x01
    Interface Version: 0x00
    Message Type: 0x80 (Response)
    Return Code: 0x00 (Ok)
}
Payload: 09080706050403020100
```

6 onMessage(...)

clientA(App1)

routingmanagerd

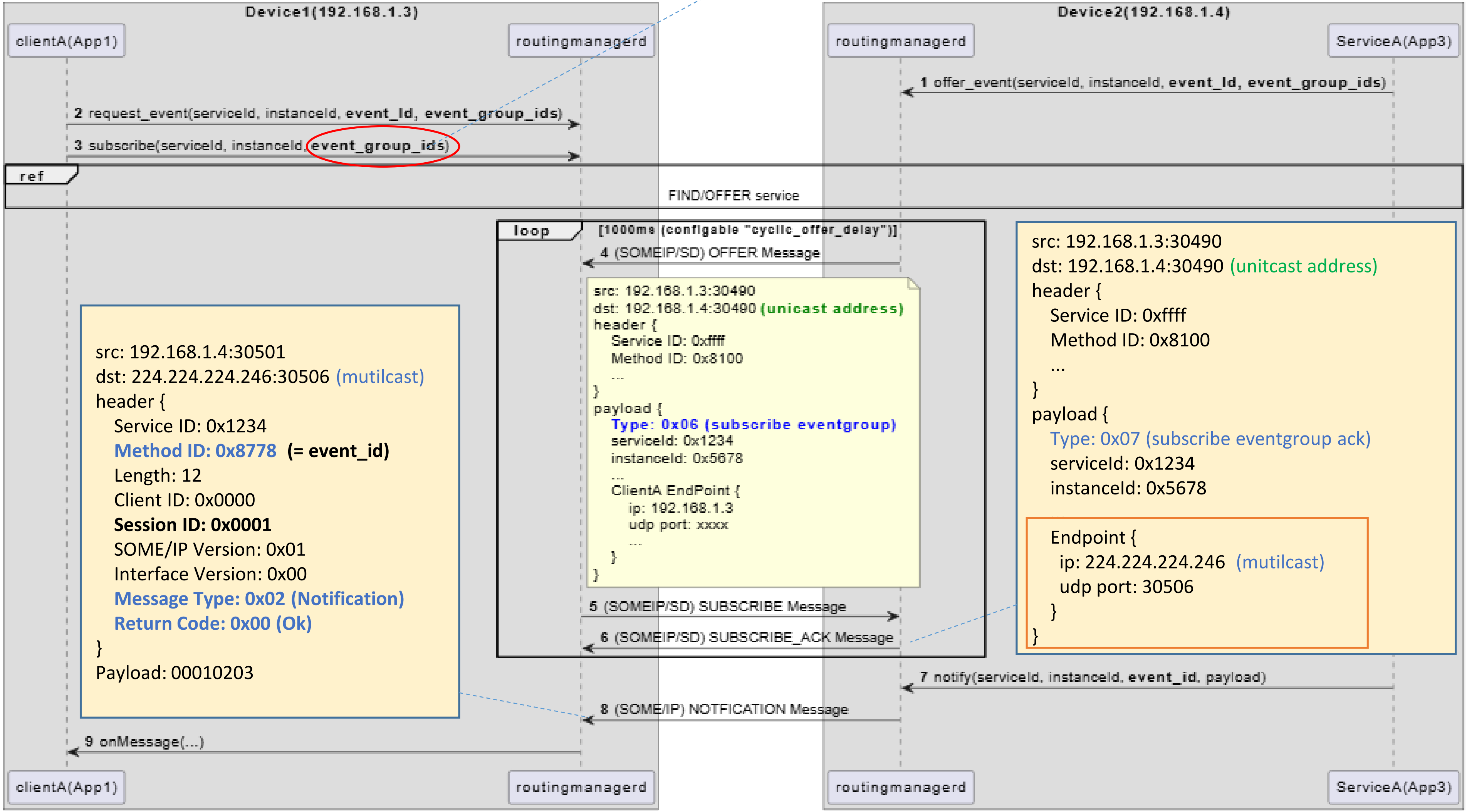
routingmanagerd

ServiceA(App3)

3. Vsomeip

- Publish/Subscribe (mutilcast)

Why subscribe event_group_id instead of event_id ?


Device1(192.168.1.3)
clientA(App1)
routingmanagerd
Device2(192.168.1.4)
routingmanagerd
ServiceA(App3)
1 offer_event(serviceId, instanceId, event_Id, event_group_ids)
2 request_event(serviceId, instanceId, event_Id, event_group_ids)
3 subscribe(serviceId, instanceId, event_group_ids)
ref
FIND/OFFER service
loop
[1000ms (configable "cyclic_offer_delay")]
4 (SOMEIP/SD) OFFER Message
5 (SOMEIP/SD) SUBSCRIBE Message
6 (SOMEIP/SD) SUBSCRIBE_ACK Message
7 notify(serviceId, instanceId, event_id, payload)
8 (SOME/IP) NOTFICATION Message
9 onMessage(...)
clientA(App1)
routingmanagerd
routingmanagerd
ServiceA(App3)

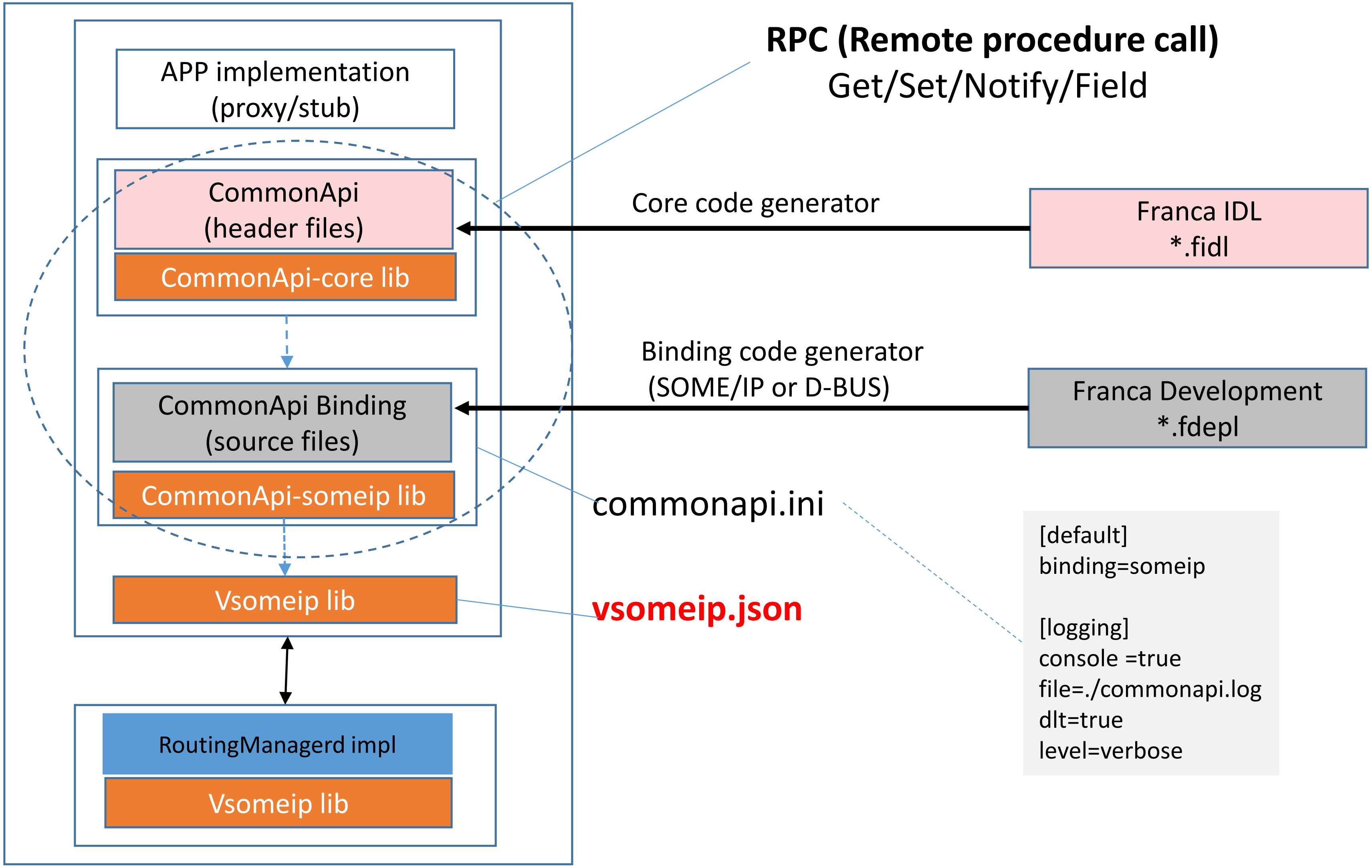
```
src: 192.168.1.3:30490
dst: 192.168.1.4:30490 (unicast address)
header {
   Service ID: 0xffff
   Method ID: 0x8100
   ...
}
payload {
   Type: 0x06 (subscribe eventgroup)
   serviceId: 0x1234
   instanceId: 0x5678
   ...
   ClientA EndPoint {
      ip: 192.168.1.3
      udp port: xxxx
      ...
   }
}
```

```
src: 192.168.1.3:30490
dst: 192.168.1.4:30490 (unitcast address)
header {
   Service ID: 0xffff
   Method ID: 0x8100
   ...
}
payload {
   Type: 0x07 (subscribe eventgroup ack)
   serviceId: 0x1234
   instanceId: 0x5678
   ...
   Endpoint {
    ip: 224.224.224.246  (mutilcast)
    udp port: 30506
   }
}
```

```
src: 192.168.1.4:30501
dst: 224.224.224.246:30506 (mutilcast)
header {
   Service ID: 0x1234
   Method ID: 0x8778  (= event_id)
   Length: 12
   Client ID: 0x0000
   Session ID: 0x0001
   SOME/IP Version: 0x01
   Interface Version: 0x00
   Message Type: 0x02 (Notification)
   Return Code: 0x00 (Ok)
}
Payload: 00010203
```

3. Vsomeip + CommonApi

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



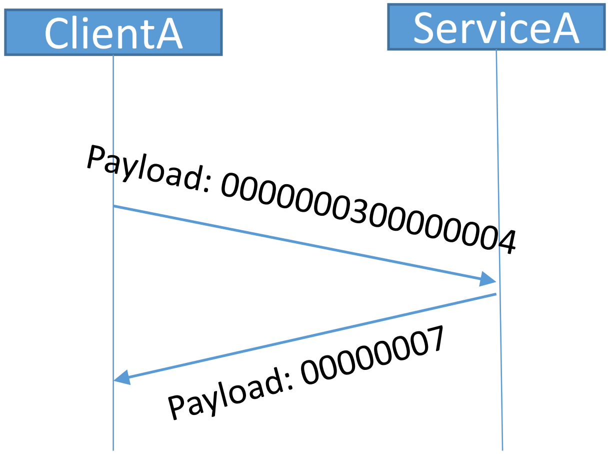
```
[default]
binding=someip

[logging]
console =true
file=./commonapi.log
dlt=true
level=verbose
```

3. 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;
myProxy->calculateSum(a, b, callStatus, sum , &info);
Printf("sum = %d", sum ); // sum = 7
```

```
…
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 );
}
```

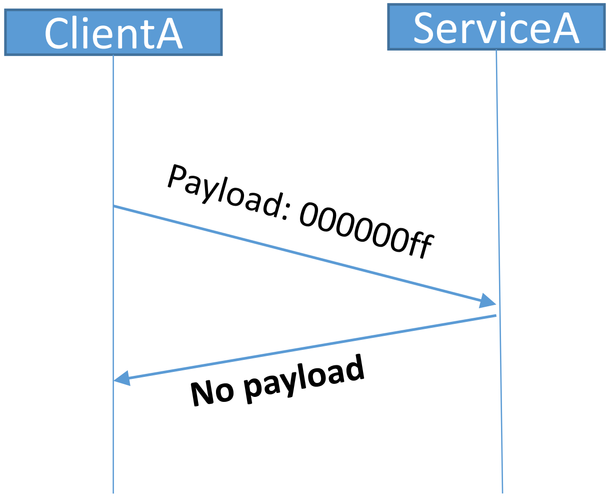
3. Vsomeip + CommonApi

➢ 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);
```

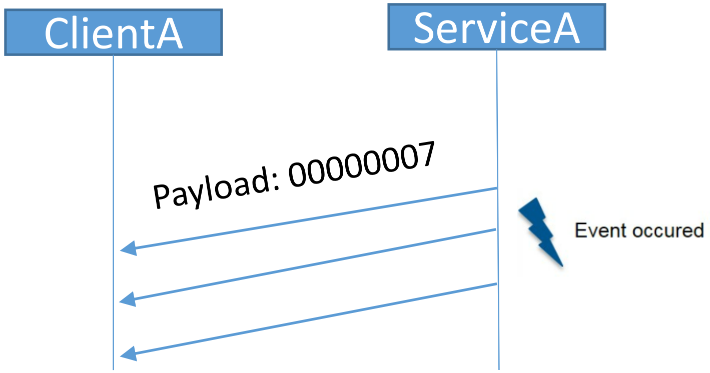
```
…
XXXStubImpl::setValue(
   const std::shared_ptr<CommonAPI::ClientId> _client,
   int32_t x, setValueReply_t _reply) {

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

3. Vsomeip + CommonApi

RPC (Remote procedure call)

❑ **NOTIFY Method (pub-sub)**

void onMyStatusChanged (int32_t myStatus);



Notification (0x02)

```
…
Int32_t x = 255;
// 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).

3. Vsomeip + CommonApi

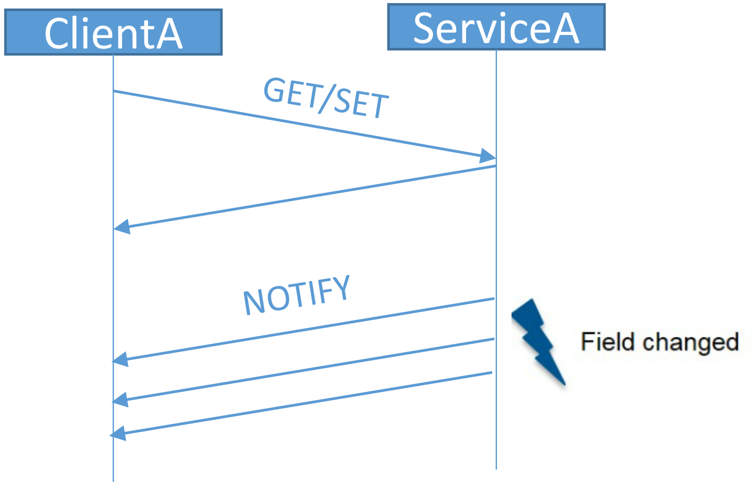
➢ RPC (Remote procedure call)

❑ **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).

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)
```



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

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