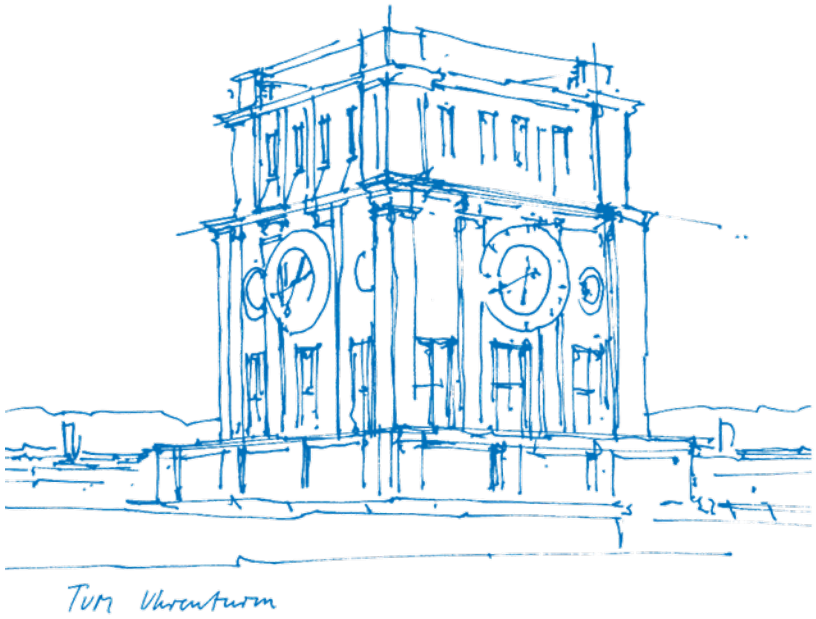
Chair of Network Architectures and Services
TUM Department of Informatics
Technical University of Munich (TUM)

Anomaly Detection for SOME/IP using Complex Event Processing

Nadine Herold, Stephan-A. Posselt, Oliver Hanka, Georg Carle

Istanbul, Turkey, April 29, 2016



Motivation

Changes in automotive and the aerospace domain

- More convenience at lower costs in the automotive and the aerospace domain lead to usage of *common* protocols, e.g. TCP/IP suite.
- New protocols used upon IP are designed and implemented.
- They offer a large *attack surface*, and cars and airplanes are valuable targets.
- The rapid development of such protocols accompany various *security related challenges*.
- The correct behavior of all components depends on the correctness of the implementation.

We propose a system for:

- offline-testing new protocol implementations in a convenient way and
- live detection of attacks in a running system.

Some-IP

Scalable Service-Oriented Middleware for IP

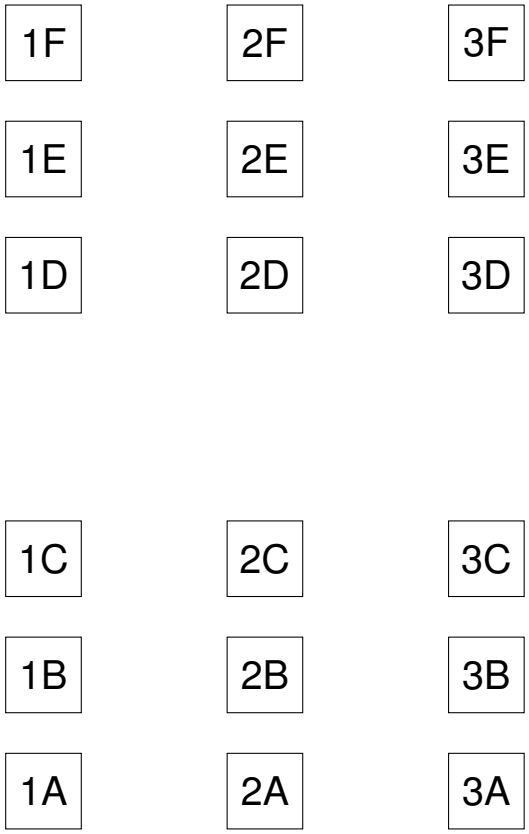
- Standardized by AUTOSAR project
- Remote procedure calls (RPC) on top of the TCP/IP protocol stack
- No built-in security measures

Possible misuse cases or *attacks*:

- Malformed packets
- Protocol violations
- System- or use case specific violations
- Timing issues

Use-Case: Cabin Data Network

Simplified Airbus Mock-Up

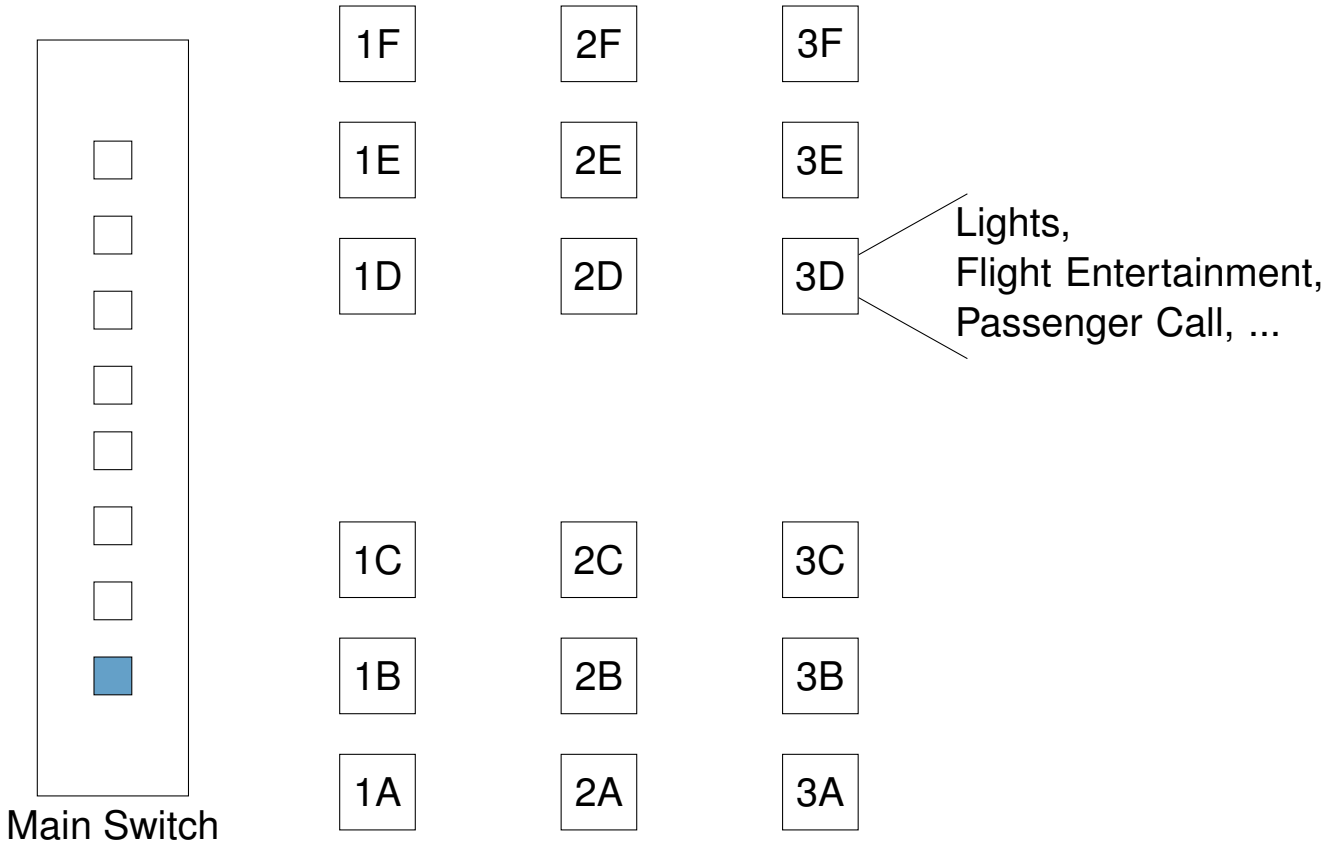


Use-Case: Cabin Data Network

Simplified Airbus Mock-Up

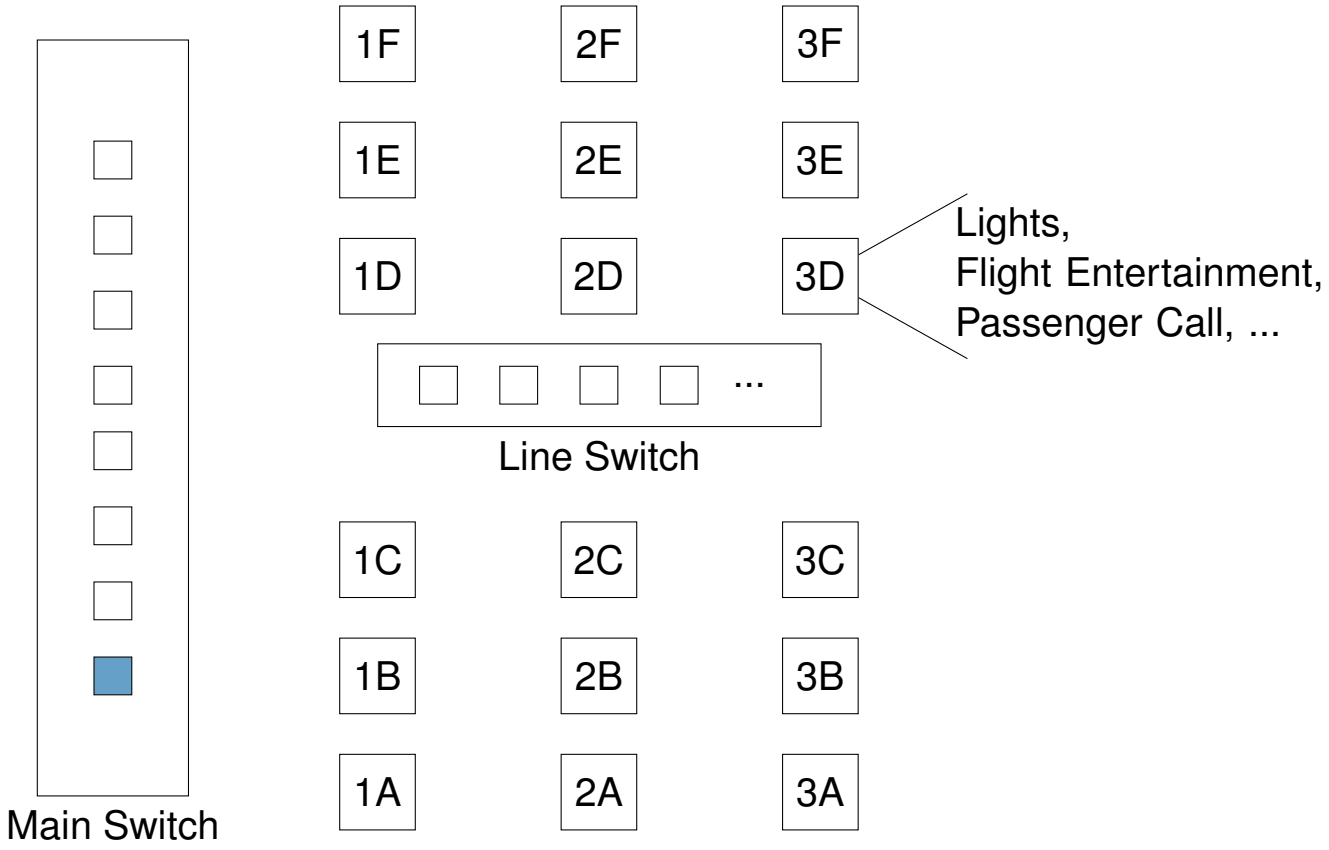
Use-Case: Cabin Data Network

Simplified Airbus Mock-Up



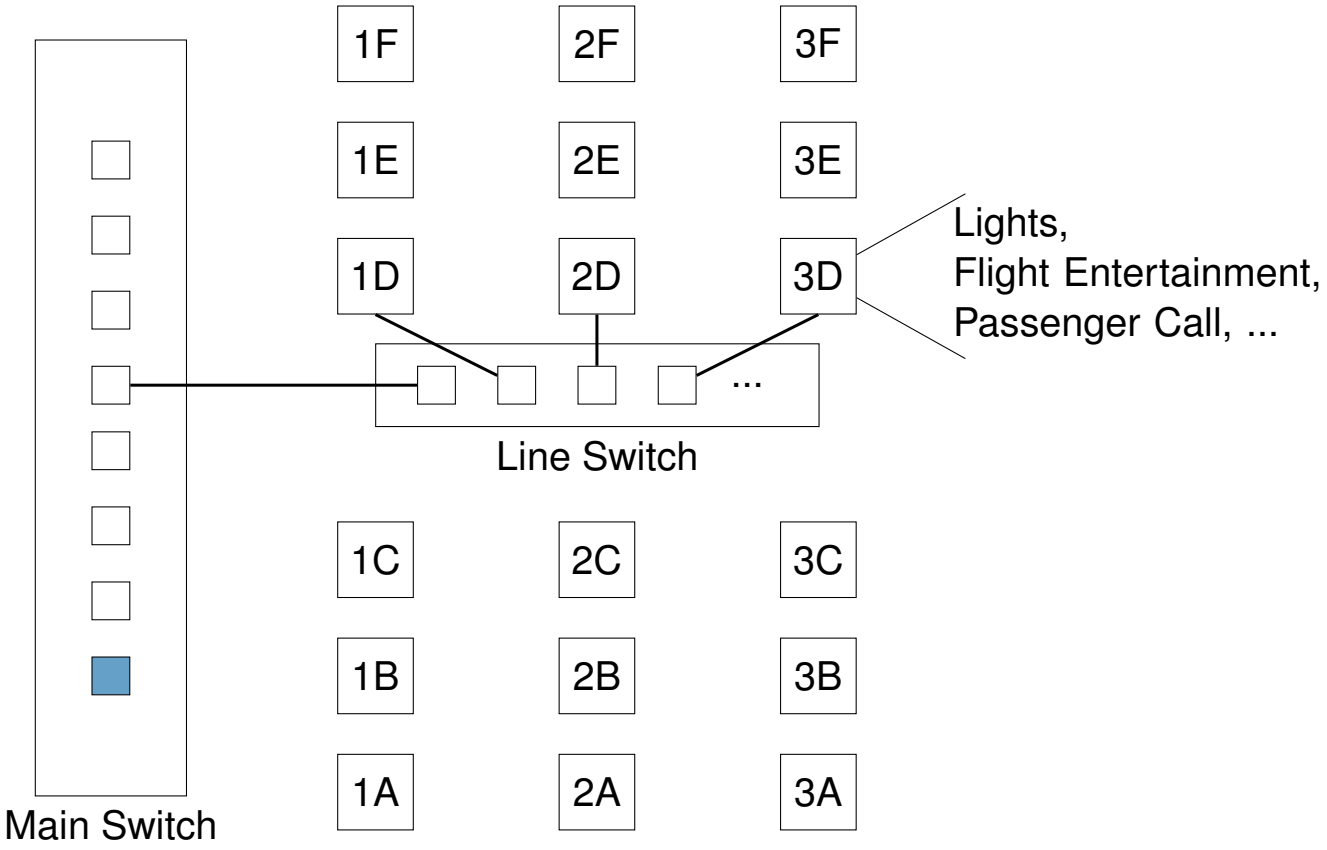
Use-Case: Cabin Data Network

Simplified Airbus Mock-Up



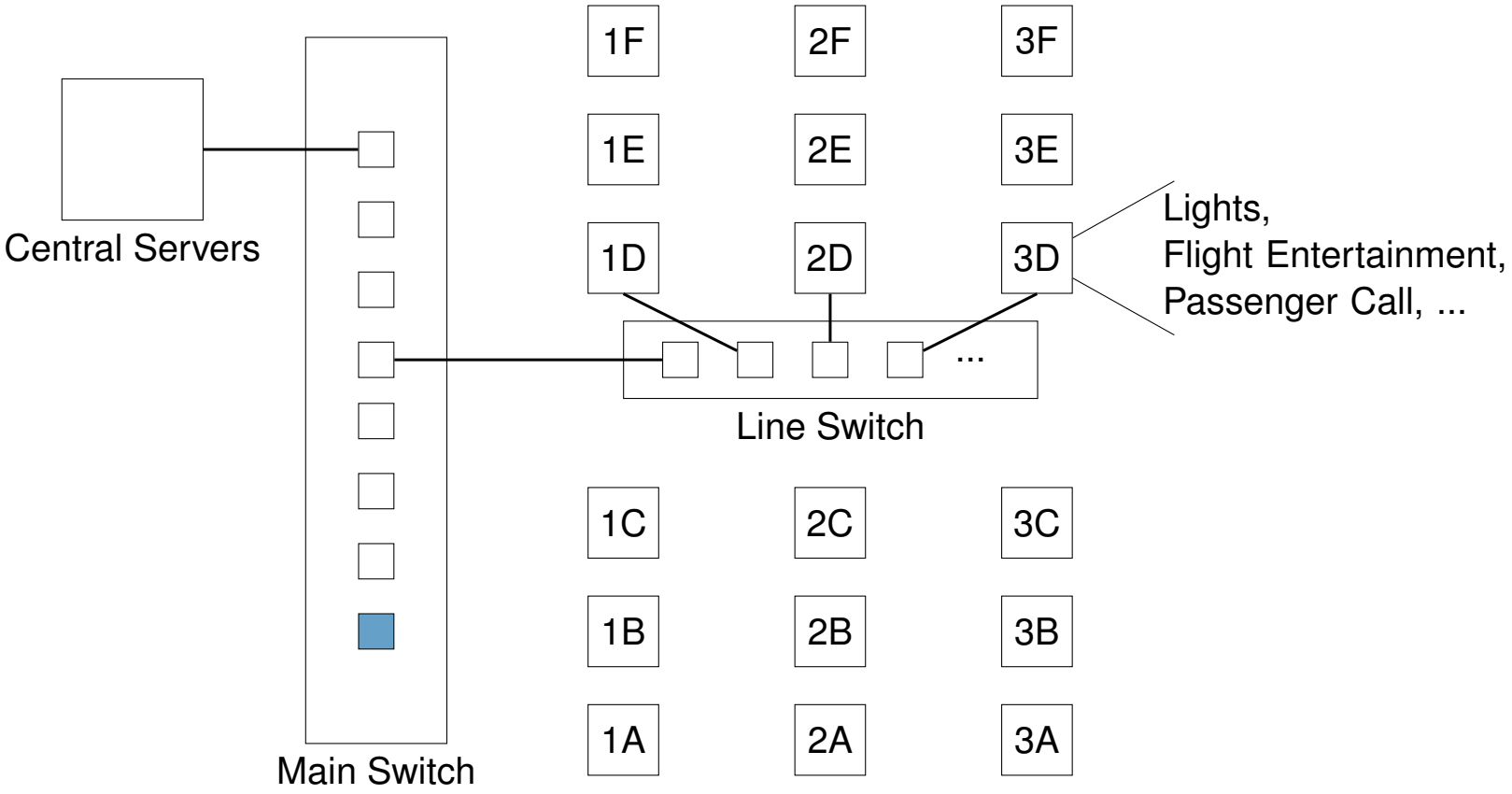
Use-Case: Cabin Data Network

Simplified Airbus Mock-Up



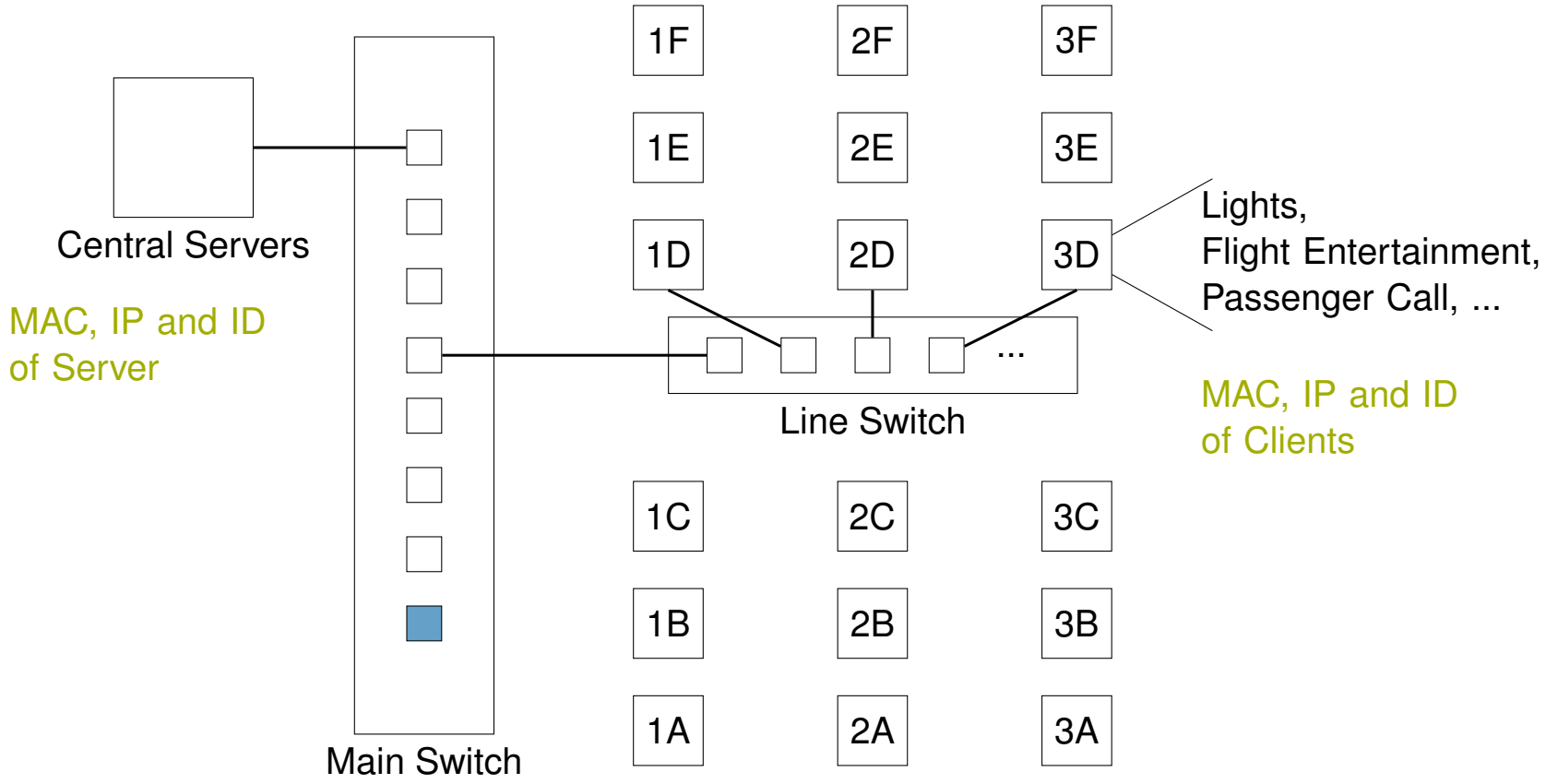
Use-Case: Cabin Data Network

Simplified Airbus Mock-Up



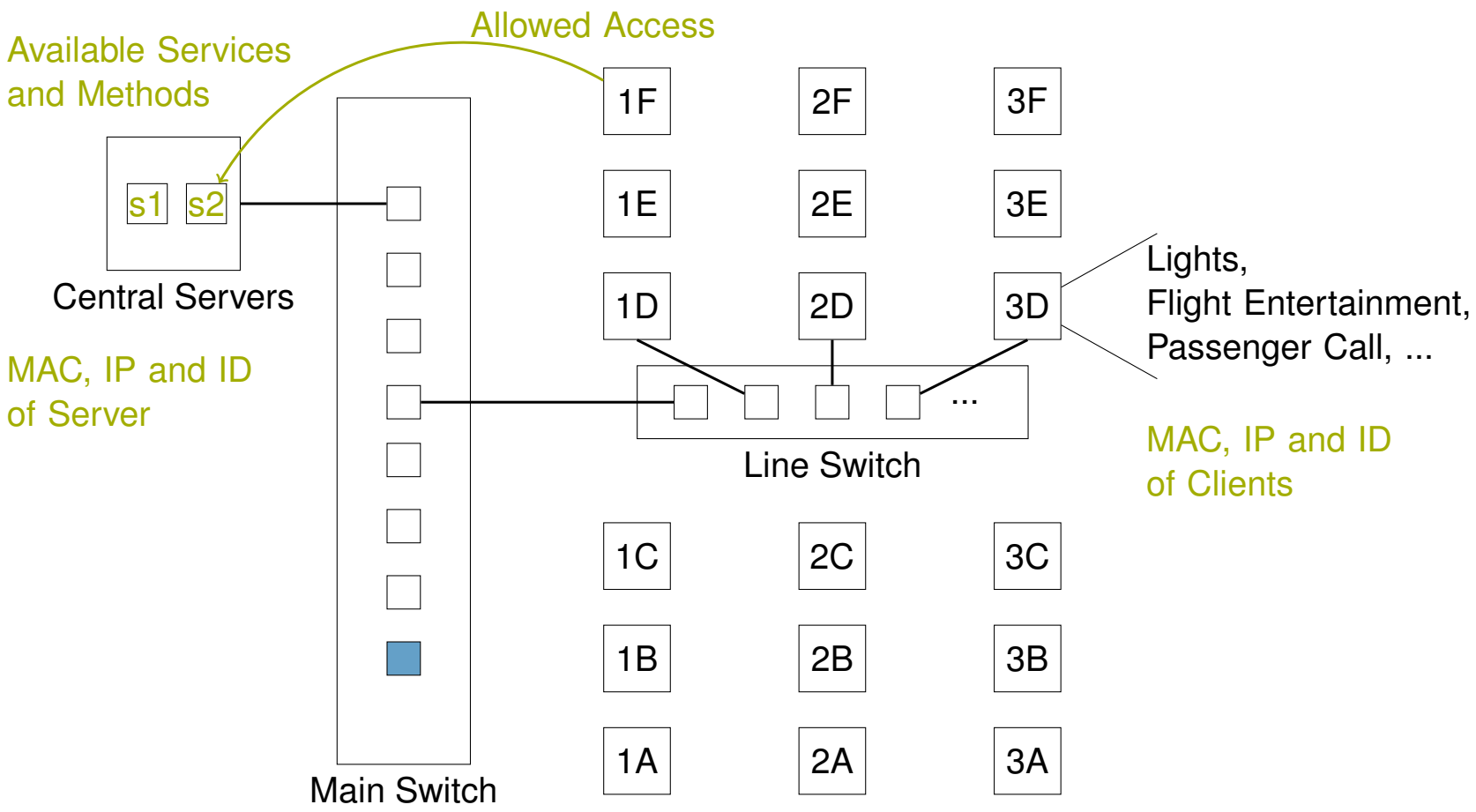
Use-Case: Cabin Data Network

System Information known Before-Hand



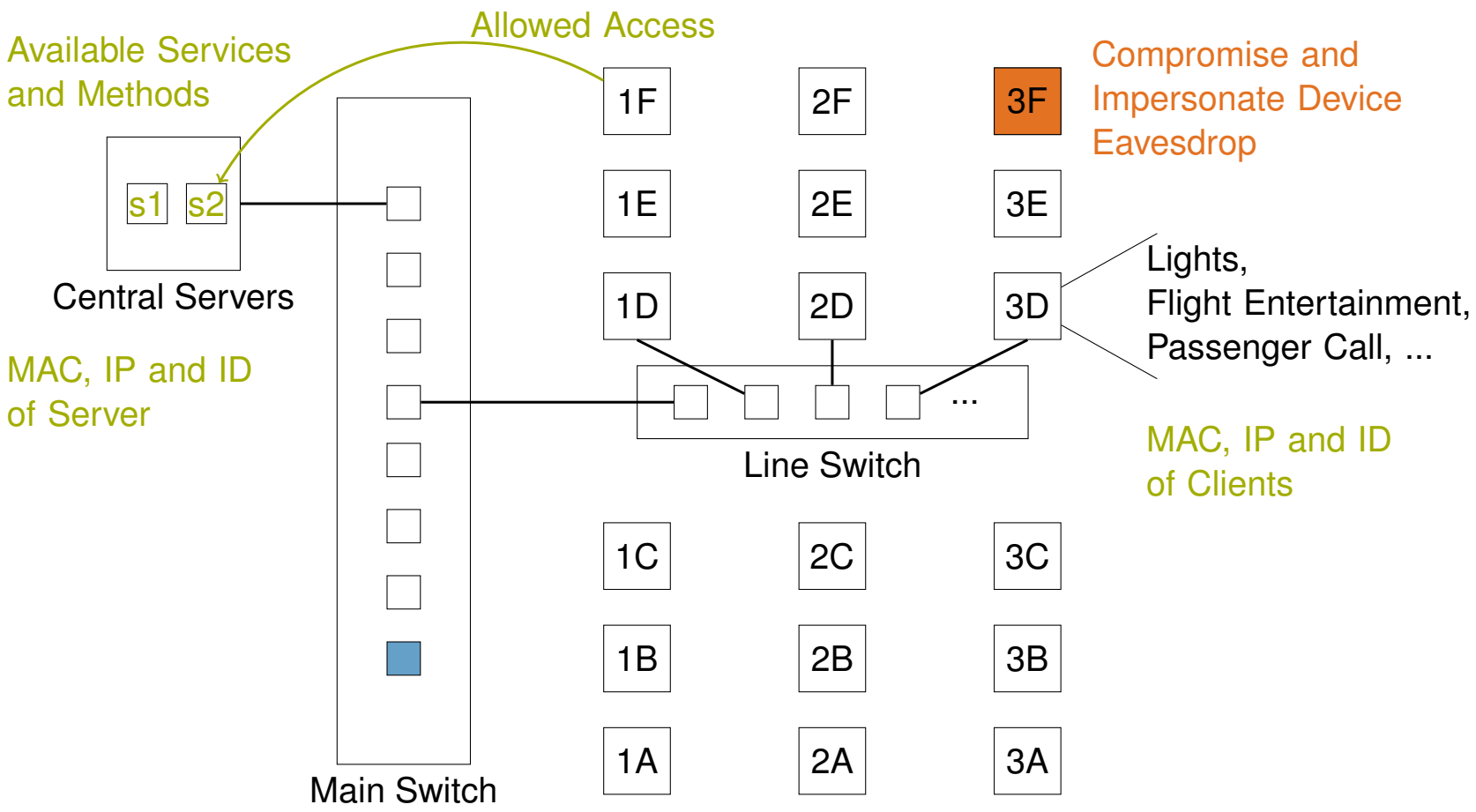
Use-Case: Cabin Data Network

System Information known Before-Hand



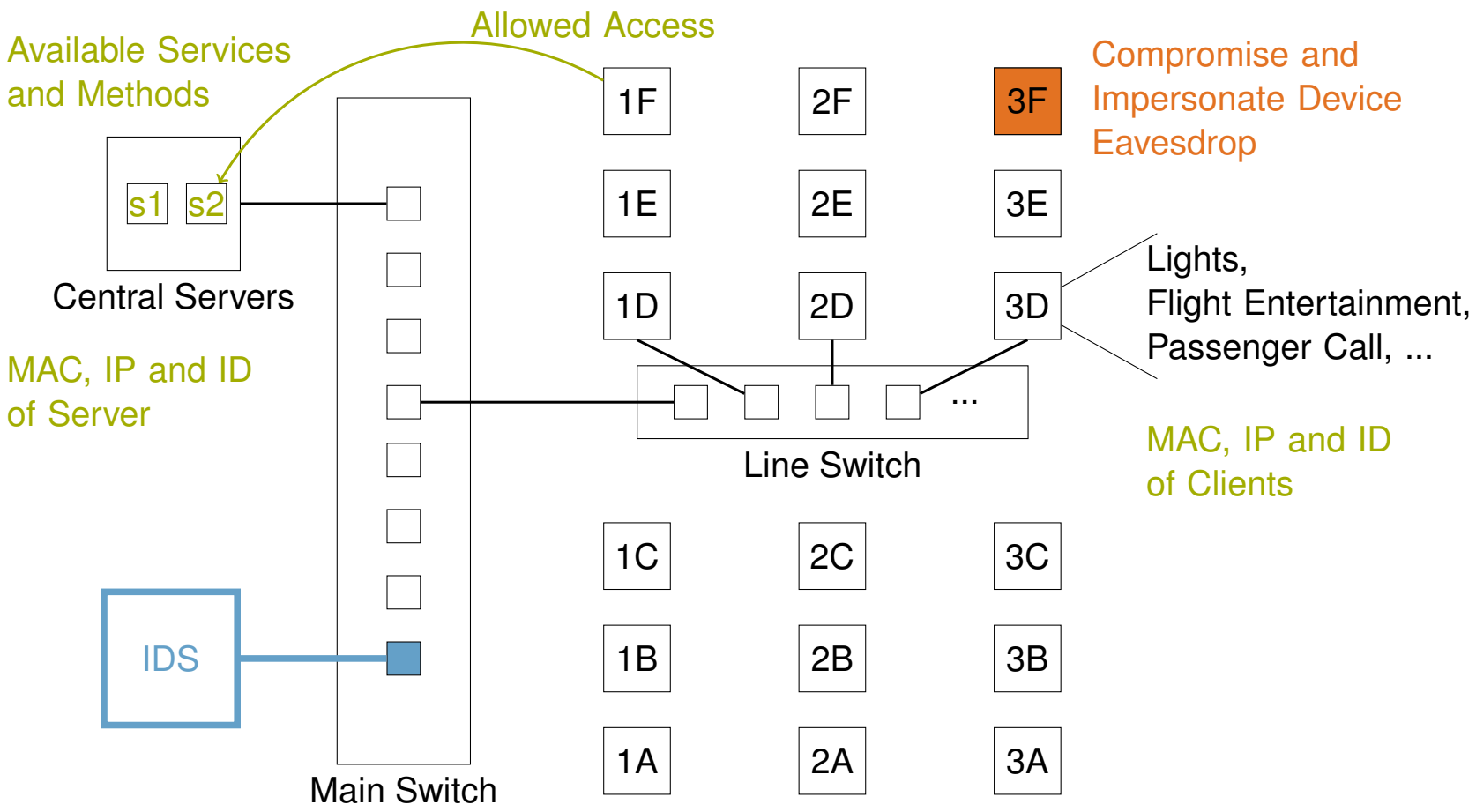
Use-Case: Cabin Data Network

Attacker Model



Use-Case: Cabin Data Network

Proposed System Placement

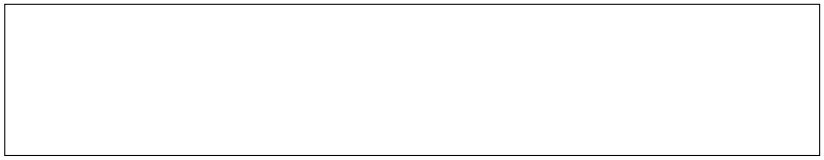


Esper: A Complex Event Processing Engine

Basic Event Flow

R1: Event is red.
R2: Two events are red.

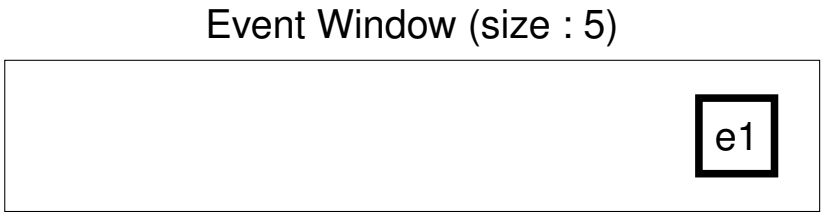
Event Window (size : 5)



Esper: A Complex Event Processing Engine

Basic Event Flow

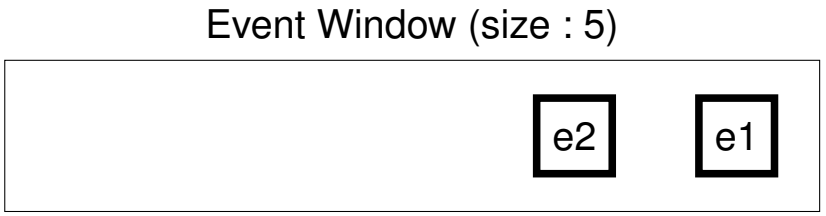
R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

Basic Event Flow

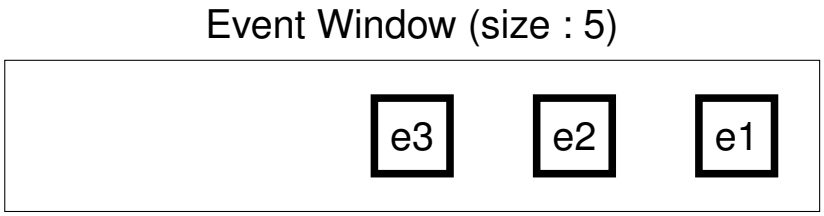
R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

Basic Event Flow

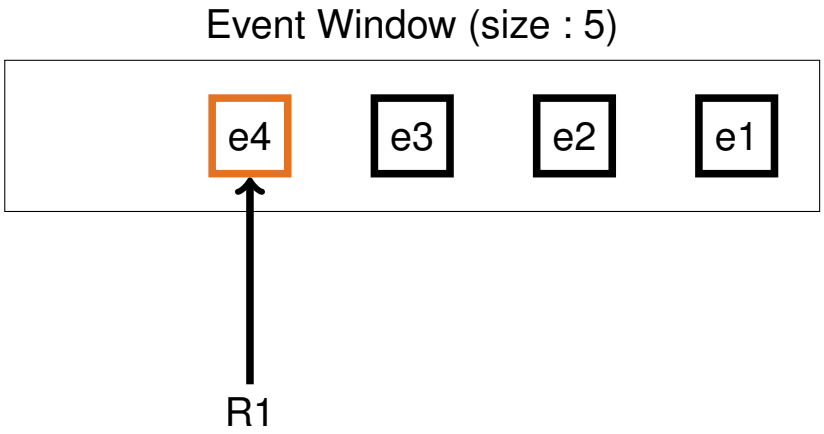
R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

Basic Event Flow

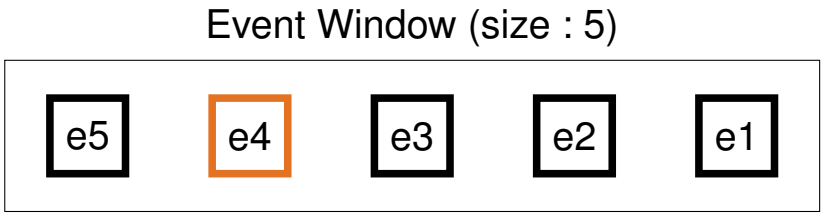
R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

Basic Event Flow

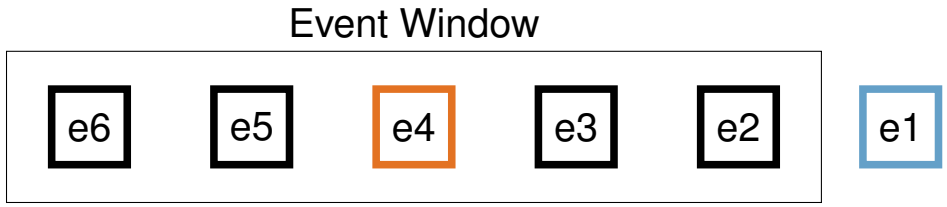
R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

Basic Event Flow

R1: Event is red.
R2: Two events are red.



Esper: A Complex Event Processing Engine

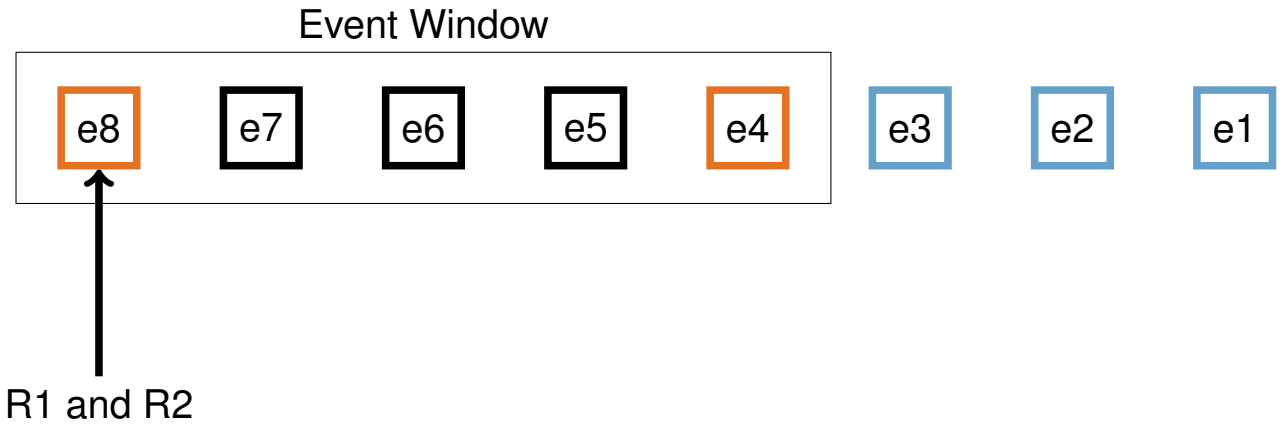
Basic Event Flow

R1: Event is red.
R2: Two events are red.

Esper: A Complex Event Processing Engine

Basic Event Flow

R1: Event is red.
R2: Two events are red.



EPL

Event Processing Language

Check for Timing Constraints:

```
1  SELECT * FROM SomeIPPacket(clientID = id, methodID = x, serviceID = y).win:length
       (1) as s1
2   WHERE NOT EXISTS
3     (SELECT * FROM SomeIPPacket(clientID = id, methodID = x, serviceID = y).win:
         length(2) as s2
4      WHERE s2.timestamp < s1.timestamp − δ)
```

EPL

Event Processing Language

Check for correct error behavior (no error is sent on a previous error):

```
1   SELECT * FROM SomeIPPacket(type = ERROR).win:length(1) s1
2    WHERE NOT EXISTS
3      (SELECT * FROM SomeIPPacket(type = REQUEST OR type = NOTIFICATION OR .type =
         REQUEST_NO_RETURN).win:length(100) s2
4       WHERE s1.serviceID = s2.serviceID
5       AND s1.methodID = s2.methodID
6       AND s1.requestID = s2.requestID
7       AND s1.srcIP = s2.dstIP
8       AND s1.dstIP = s2.srcIP
9       AND s1.srcMAC = s2.dstMAC
10      AND s1.dstMAC = s2.srcMAC
11      AND s1.srcPort = s2.dstPort
12      AND s1.dstPort = s2.srcPort
13      AND s1.timestamp > s2.timestamp
14      AND s2.timestamp < s1.timestamp + δ)
```

Results

Implemented and Tested Rules

- Correct Error Behavior
  - No error is sent on another error
  - No error is sent on event type NOTIFICATION
- Check for missing messages
  - No response is missing
  - No request is missing
- Disturbed Timing (given times for notification intervals)
- Malformed packets (wrong interface)
- No spoofed Client ID

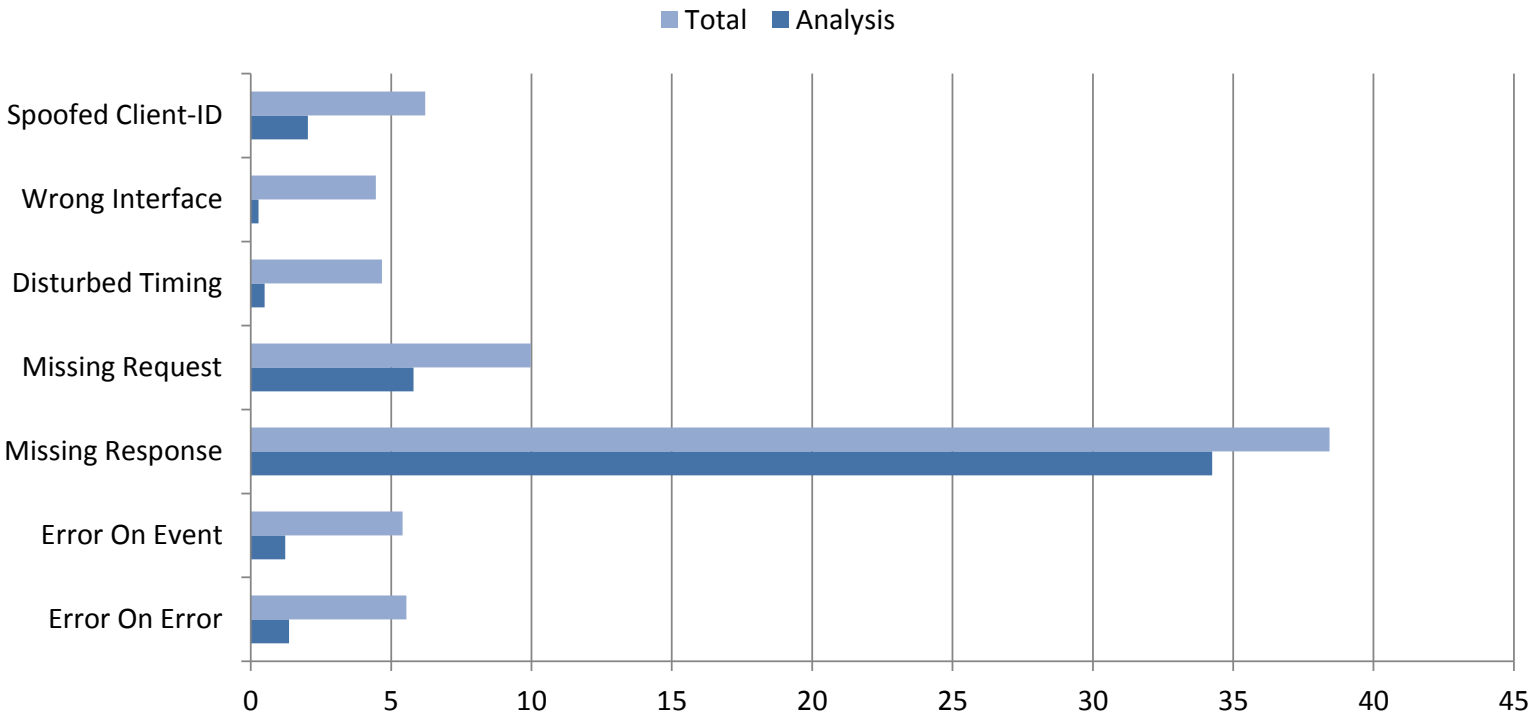
Results

Time Comparison in Seconds - Single Rules

For evaluation, we generated a libpcap dump file containing 12.000 attacks, with a size of 122.4MB and containing around 1.49 million packets. The pure deserialization time is 4.18 seconds.

Results

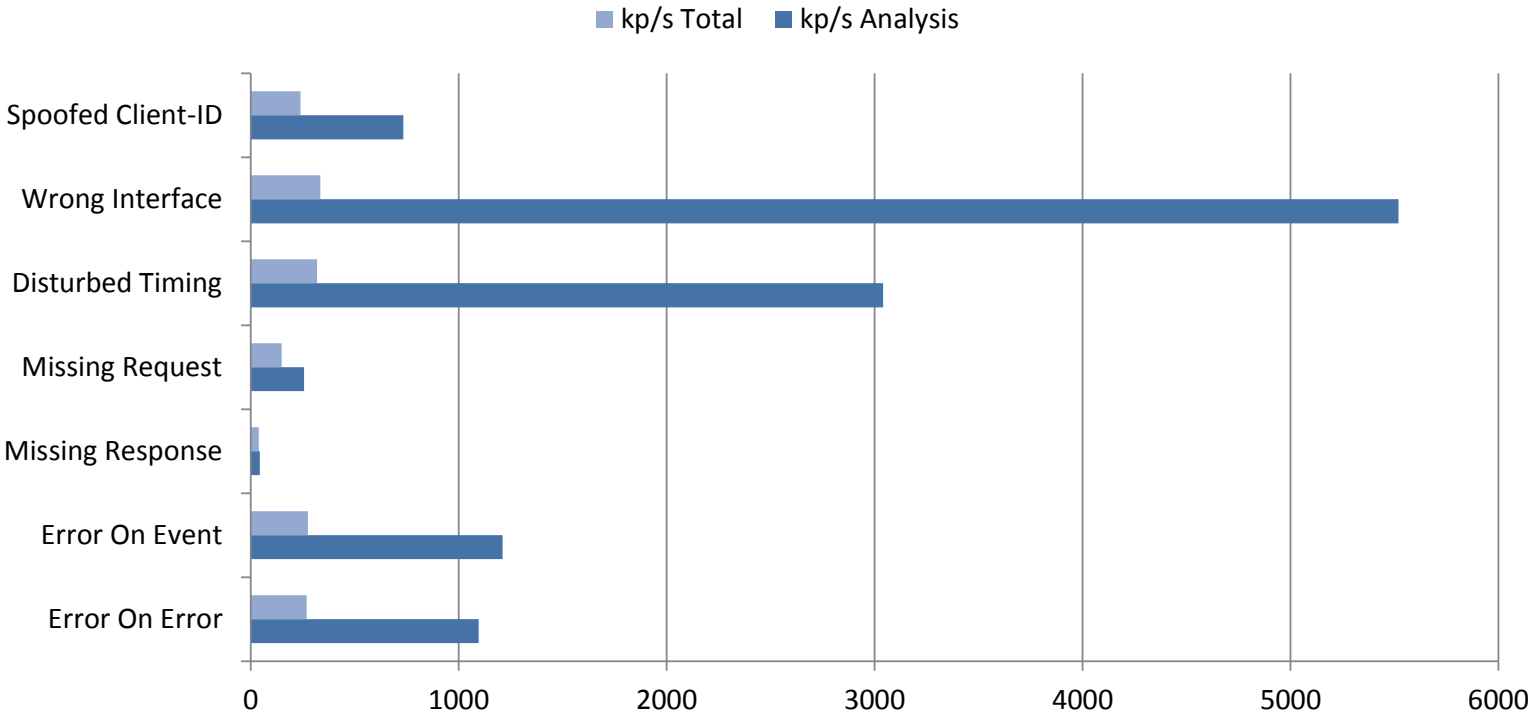
Time Comparison in Seconds - Single Rules



	Error On Error	Error On Event	Missing Response	Missing Request	Disturbed Timing	Wrong Interface	Spoofed Client-ID
Total	5,54	5,41	38,43	9,98	4,67	4,45	6,21
Analysis	1,36	1,23	34,25	5,8	0,49	0,27	2,03

Results

Time Comparison in 1000 Packets per Second - Single Rules

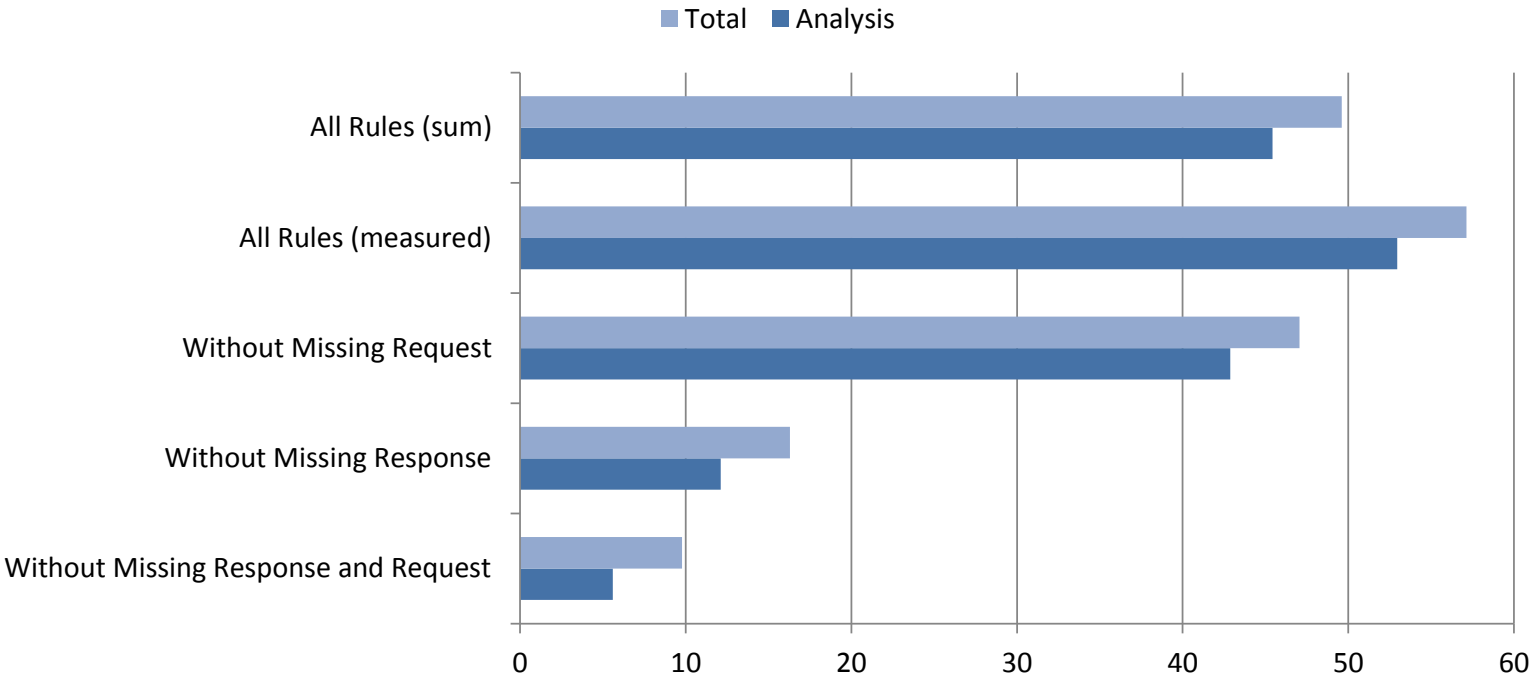


	Error On Error	Error On Event	Missing Response	Missing Request	Disturbed Timing	Wrong Interface	Spoofed Client-ID
kp/s Total	269	275	39	149	319	335	240
kp/s Analysis	1096	1211	44	257	3041	5519	734

1000 Packtes per Second

Results

Time Comparison in Second - Multiple Rules

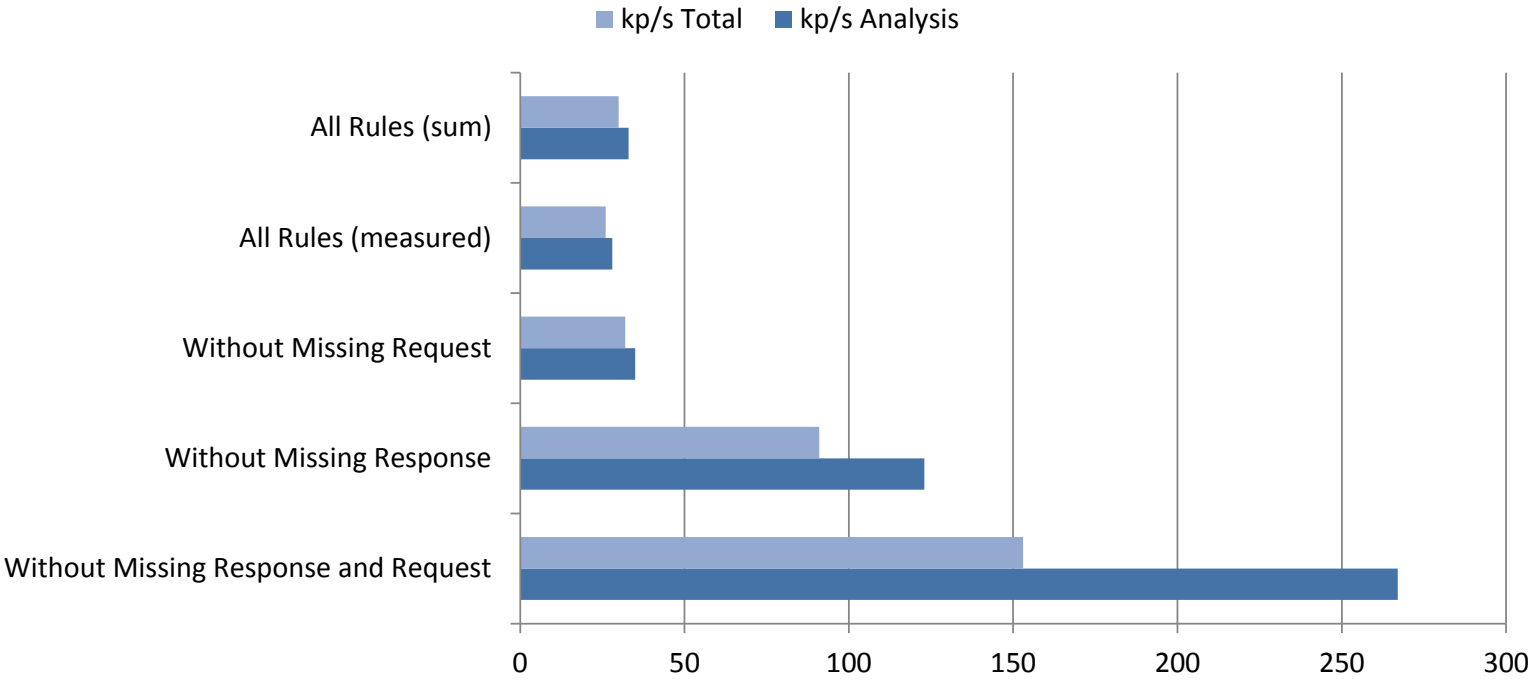


	Without Missing Response and Request	Without Missing Response	Without Missing Request	All Rules (measured)	All Rules (sum)
Total	9,77	16,29	47,05	57,13	49,61
Analysis	5,59	12,11	42,87	52,95	45,43

Time in Seonds

Results

Time Comparison in 1000 Packets per Second - Multiple Rules



	Without Missing Response and Request	Without Missing Response	Without Missing Request	All Rules (measured)	All Rules (sum)
kp/s Total	153	91	32	26	30
kp/s Analysis	267	123	35	28	33

1000 Packets per Seond

Conclusion

- Identified possible attacks on SOME/IP
- Demonstrated that the simple, SQL-like language EPL can be used to express non-trivial checks on a stream of network packets
- Showed that the system is usable for testing implementations for rapid prototyping.
- Showed that the implemented system can only run a sub-set of rules for the aircraft cabin network at line rate

Questions?

Nadine Herold <herold@net.in.tum.de>

Source Code Available:

https://github.com/Egomania/SOME-IP_Generator

https://github.com/Egomania/SOME-IP_Analyzer

Feedback, experiences and improvements are welcome!

Acknowledgments: This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS0145, project SURF.

Back-Up

Malformed Packets

```
1  SELECT * FROM SomeIPPacket.win:length(1)
2  WHERE interfaceVersion != INTERFACE
```

Check of changed Client ID/IP assignment

```
1   ON SomeIPPacket s MERGE clientMappingIP cm
2    WHERE (s.srcIP in (select client_ip from clientMappingIP)
3    AND (s.srcIP in (clientIPs))
4    AND (s.srcIP = cm.client_ip AND s.clientID != cm.client_id))
5   WHEN MATCHED THEN
6    UPDATE SET cm.client_id = setClientID(s)
7   WHEN NOT MATCHED
8    AND s.srcIP not in (select client_ip from clientMappingIP)
9    AND s.srcIP in (clientIPs) THEN
10    INSERT into clientMappingIP select s.srcIP as client_ip, s.clientID as
        client_id
```

Check for correct error behavior

```
1   SELECT * FROM SomeIPPacket(type = ERROR).win:length(1) s1
2    WHERE NOT EXISTS
3      (SELECT * FROM SomeIPPacket(type = REQUEST OR type = NOTIFICATION OR .type =
         REQUEST_NO_RETURN).win:length(100) s2
4       WHERE s1.serviceID = s2.serviceID
5       AND s1.methodID = s2.methodID
6       AND s1.requestID = s2.requestID
7       AND s1.srcIP = s2.dstIP
8       AND s1.dstIP = s2.srcIP
9       AND s1.srcMAC = s2.dstMAC
10      AND s1.dstMAC = s2.srcMAC
11      AND s1.srcPort = s2.dstPort
12      AND s1.dstPort = s2.srcPort
13      AND s1.timestamp > s2.timestamp
14      AND s2.timestamp < s1.timestamp + δ)
```

Check for missing request

```
1   SELECT * FROM SomeIPPacket(type = RESPONSE).win:length(1) s1
2   WHERE NOT EXISTS (
3     SELECT * FROM SomeIPPacket(type = REQUEST).win:length(100) s2
4      WHERE (s1 corresponds to s2)
5      AND s1.timestamp > s2.timestamp
6      AND s2.timestamp < s1.timestamp + δ))
7     OR
8      ((SELECT count(*) from SomeIPPacket(type = REQUEST).win:length(50) s2
9       WHERE (s1 corresponds to s2)
10      AND s1.timestamp > s2.timestamp
11      AND s2.timestamp < s1.timestamp + δ)
12     =
13     (SELECT count(*) from SomeIPPacket(type = ERROR or type = RESPONSE).win:length
          (50) s2
14      WHERE (s1 equals s2)
15      AND s1.timestamp > s2.timestamp
16      AND s2.timestamp < s1.timestamp + δ
17      AND s2.timestamp > minValue )))
```

Check for missing request (Helper Query)

```
1  ON SomeIPPacket (type = RESPONSE) as s1
2   SET minValue = (
3    SELECT min(s.timestamp) FROM SomeIPPacket(type = REQUEST).win:length(100) s
4    WHERE (s1 equals s2)
5    AND s1.timestamp > s.timestamp
6    AND s.timestamp < s1.timestamp + δ
```

Check for Missing Response

```
1   SELECT * FROM
2     SomeIPPacket.win:length(1) s1,
3     SomeIPPacket.win:length(1) s2,
4     SomeIPPacket(type = REQUEST).win:length(100) s3
5   WHERE s1.timestamp > s2.timestamp
6     AND s3.timestamp < (s1.timestamp − δ)
7     AND s3.timestamp > (s2.timestamp − δ)
8     AND NOT EXISTS
9       (SELECT * FROM SomeIPPacket(type = RESPONSE OR type = ERROR).win:length(50) s4
10      WHERE (s3 corresponds to s4)
11      AND s3.timestamp < s4.timestamp)
```