Verification of the Session Management Protocol

A Formal Methods Case Study

Karl Palmskog

School of Computer Science and Communication
Royal Institute of Technology

2006-11-02

Goals

- Examplify formal methods for verification of software
- Report on the verification of the Session Management Protocol
- Highlight the view of concurrency as interaction

Formal Methods: An Example

The mutual exclusion problem for the concurrent processes P_0 and P_1 using shared memory:

- Each process wants to access a shared resource, but both processes must not get access simultaneously
- A process using the resource is in its “critical section”

Formal Methods: An Example

Peterson's algorithm for mutual exclusion

bool $b_0 := \mathit{false}$; **bool** $b_1 := \mathit{false}$; **int** $k := 0$;

P_0:
while *true* **do**
  $\langle$noncritical section$\rangle$;
  $b_0 := \mathit{true}$;
  $k := 1$;
  $\mathit{await}(\neg b_1 \vee k = 1)$;
  $\langle$critical section$\rangle$;
  $b_0 := \mathit{false}$;
end while

P_1:
while *true* **do**
  $\langle$noncritical section$\rangle$;
  $b_1 := \mathit{true}$;
  $k := 0$;
  $\mathit{await}(\neg b_0 \vee k = 0)$;
  $\langle$critical section$\rangle$;
  $b_1 := \mathit{false}$;
end while

Formal Methods: An Example

How can we convince ourselves that this algorithm works?

- By inspection?
- By implementing and testing it?
- By proving it correct?

Formal Methods: An Example

General formal methods methodology

1. Understand the program
2. Model the program in a suitable formalism
3. Specify the correctness of the program
4. Prove that the model satisfies the specification

Formal Methods: An Example

Peterson's algorithm as a communication protocol

- P_0 and P_1 exchange messages with a memory process P_m
- Variable names are message types
- Values are message content
- Writing a variable means sending a message to P_m
- Reading a variable means receiving a message from P_m

Formal Methods: An Example

Promela model

```
mtype = {b0,b1,k};

bool proc0InCrit = false;
bool proc1InCrit = false;

chan mem0 = [0] of {mtype,bit};
chan mem1 = [0] of {mtype,bit};

run Memory(mem0, mem1, false, false, 0);
run Process0(mem0);
run Process1(mem1);
```

Formal Methods: An Example

```
proctype Process0(chan mem) {
BEGIN:
  mem!b0,true; mem!k,1;
  do
    :: mem?b1,false; break;
    :: mem?b1,true;
    :: mem?k,0; break;
    :: mem?k,1;
  od;
  proc0InCrit = true;
  proc0InCrit = false;
  mem!b0,false;
  goto BEGIN;
}
```

```
proctype Process1(chan mem) {
BEGIN:
  mem!b1,true; mem!k,0;
  do
    :: mem?b0,false; break;
    :: mem?b0,true;
    :: mem?k,0;
    :: mem?k,1; break;
  od;
  proc1InCrit = true;
  proc1InCrit = false;
  mem!b1,false;
  goto BEGIN;
}
```

Formal Methods: An Example

Correctness of Peterson's algorithm

"For all executions, there are no states where both `proc0InCrit` and `proc1InCrit` have assumed the value *true*."

In Linear Temporal Logic:

$$\Box\,(\neg(p0c \wedge p1c))$$

where

#define p0c proc0InCrit $==$ **true**
#define p1c proc1InCrit $==$ **true**

Session Layer Resurgence

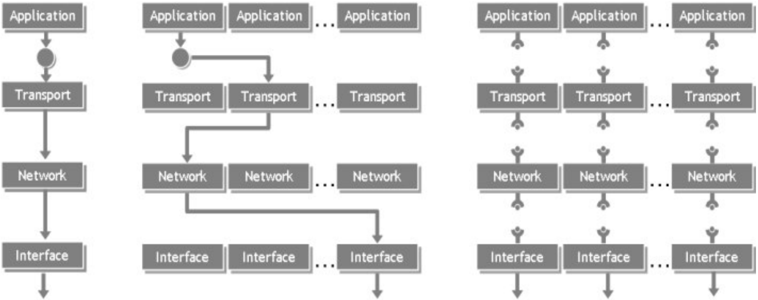
Problem situation

- Demand for new network services
- Aging Internet architecture
- Need to handle mobility and nomadicity
- Lots of extensions of TCP/IP: MIP, HIP, IPSec, ...

Proposed solution

- Adopt a more flexible view of the protocol stack
- Introduce new functionality at the session layer
- Use event-driven reconfiguration and state management

Session Layer Resurgence

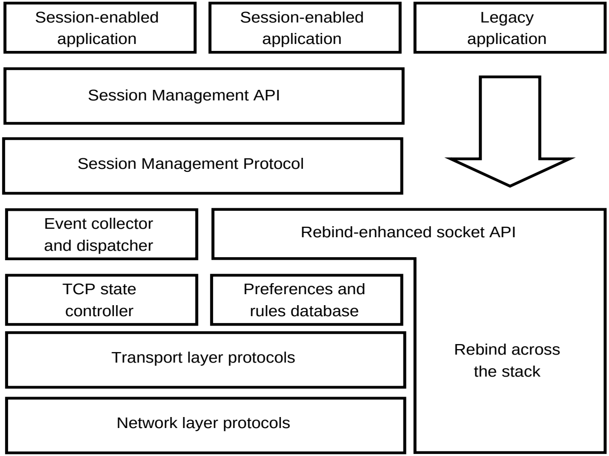


Session Layer Resurgence

Session layer components

- Event collector/dispatcher
- Preferences/rules database
- Socket rebind extension
- Session API
- TCP state controller
- Session Management Protocol (SMP)

Session Layer Resurgence



Session Layer Resurgence

Session Management Protocol

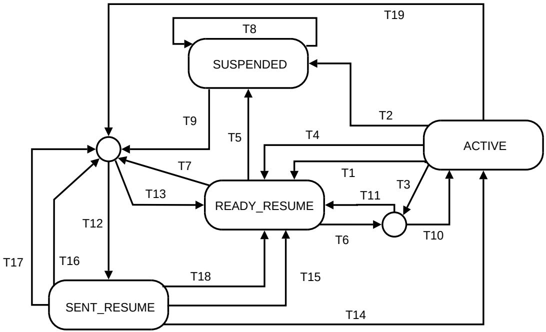
- Data integrity for sessions
- Keep track of communication state
- Send and and receive context updates

Session Layer Resurgence

SMP channels and message types

- Data channel
  - *data* — application data
  - *checkpoint* — communication state data
- Control channel
  - *resume* — request session resumption
  - *resume_ok* — confirm session resumption
  - *resume_denied* — deny session resumption
  - *suspend* — sender has suspended

State machine



T1: Network lost
T2: User suspends; send suspend
T3: Received resume; rebind
T4: Received suspend
T5: User suspends
T6: Received resume
T7: Network changed
T8: Received resume; send resume_denied
T9: User resumes
T10: Sent resume_ok; rollback
T11: Failed to send resume_ok
T12: Sent resume
T13: Failed to send resume
T14: Received resume_ok
T15: Received resume_denied
T16: Network changed; rebind
T17: Received resume; initiator
T18: Received resume; not initiator
T19: Network lost; change interface

Verification of SMP

Starting point

- Verify the checkpoint mechanism
- Lets endpoints know where to resume
- Limited scope, well-defined protocol
- Important for the correctness of SMP

Verification of SMP

Prerequisites

A, B: network endpoints
S_A, S_B: sequences of words of data
S_A^i: the ith word of a sequence

- Goal for A: transfer all words in S_A to B, in order
- Goal for B: transfer all words in S_B to A, in order

Service provisions

The purpose is to let A and B continually agree on at least one tuple $\langle i, j \rangle$, such that:

- A has received $S_B^0, S_B^1, \ldots, S_B^{j-1}$ properly
- B has received $S_A^0, S_A^1, \ldots, S_A^{i-1}$ properly

Verification of SMP

Environmental assumptions

- Executed in the context of an established session
- Endpoints use buffered, reliable data channels
- Disconnection is not possible

Procedure rules

- Same for both endpoints
- Maintain acknowledged and pending checkpoints/tuples
- After filling up the buffer, create a new checkpoint
- Send *checkpoint* message with checkpoint id and number of bytes sent/received
- Do not create checkpoints until a reply has been received
- Update checkpoint definition using reply data

Verification of SMP

Safety specification

"The endpoints always have a checkpoint in common":

$$\Box((ak \rightarrow (akSn \wedge akRc)) \wedge (akPn \rightarrow (akPnSn \wedge akPnRc)) \wedge (pnAk \rightarrow (pnAkSn \wedge pnAkRc)) \wedge ((ak \wedge \neg akPn \wedge \neg pnAk) \vee (\neg ak \wedge akPn \wedge \neg pnAk) \vee (\neg ak \wedge \neg akPn \wedge pnAk)))$$

Liveness specification

"Endpoints always eventually reach a state from which they can receive and send data":

$$(\Box \Diamond inAct) \wedge (\Box \Diamond ninAct)$$

PROMELA model

```
mtype = {data,cp};

typedef dataMsg {
  mtype type;
  byte cpId;
  byte cpSent;
  byte cpRecd;
}

chan point1Recv = [queueSize] of {dataMsg};
chan point2Recv = [queueSize] of {dataMsg};

run Endpoint(point1Recv, point2Recv, 0);
run Endpoint(point2Recv, point1Recv, 1);
```

Verification of SMP

Correcting the protocol

- Only the connection initiator can send *checkpoint* requests
- Needs to know session data buffer size of peer
- Only one stream position field in *checkpoint* message

Verification results

- Exhaustive verification with partial-order reduction
- No counterexamples found
- Without compression, would use 10-20 GB of memory

Verification of SMP

State machine correctness

- Safety: if a session is resumed, it is resumed properly
- Liveness: there are no deadlocks

State machine model

- Add control channels and states to checkpoint protocol model
- Use PROMELA's channel over channel feature for mobility
- Protocol changes during rollback due to checkpoint error

Verification results

- Exhaustively verified for some parameters
- Many partial state-space searches

Conclusions

Verification of SMP

- Unambiguous specification of the protocol
- Detection and correction of a design error
- Better understanding of the session layer

Spin

- Mature and very powerful tool
- Can be used by non-experts...
- ...but does not provide “push-the-button” verification

Formal methods

- Not just for researchers
- Should be integrated in development to increase reliability

Future Work

- Implement changes and test them
- Verify other parts of the session layer design
- Investigate SMP/TCP interaction
- Proceed to the next step in industrial applications of formal methods

"Every protocol should be considered incorrect until the opposite is proven."
—*Gerard J. Holzmann, author of* SPIN

More Information

K. Palmskog. Verification of the Session Management Protocol. Master's thesis, KTH, 2006.
http://www.palmskog.net/exjobb

Y. Ismailov, K. Palmskog, P. Arvidsson, M. Widell and Y. Wang. Session Layer Resurgence: Towards Mobile, Disconnection- and Delay-tolerant Communication. In *Proceedings of the 4th European Conference on Universal Multiservice Networks (ECUMN'2007)*, February 2007.
http://www.irit.fr/ecumn07