Split, Send, Reassemble:

A Formal Specification of a CAN Bus Protocol Stack

Rob van Glabbeek and Peter Höfner

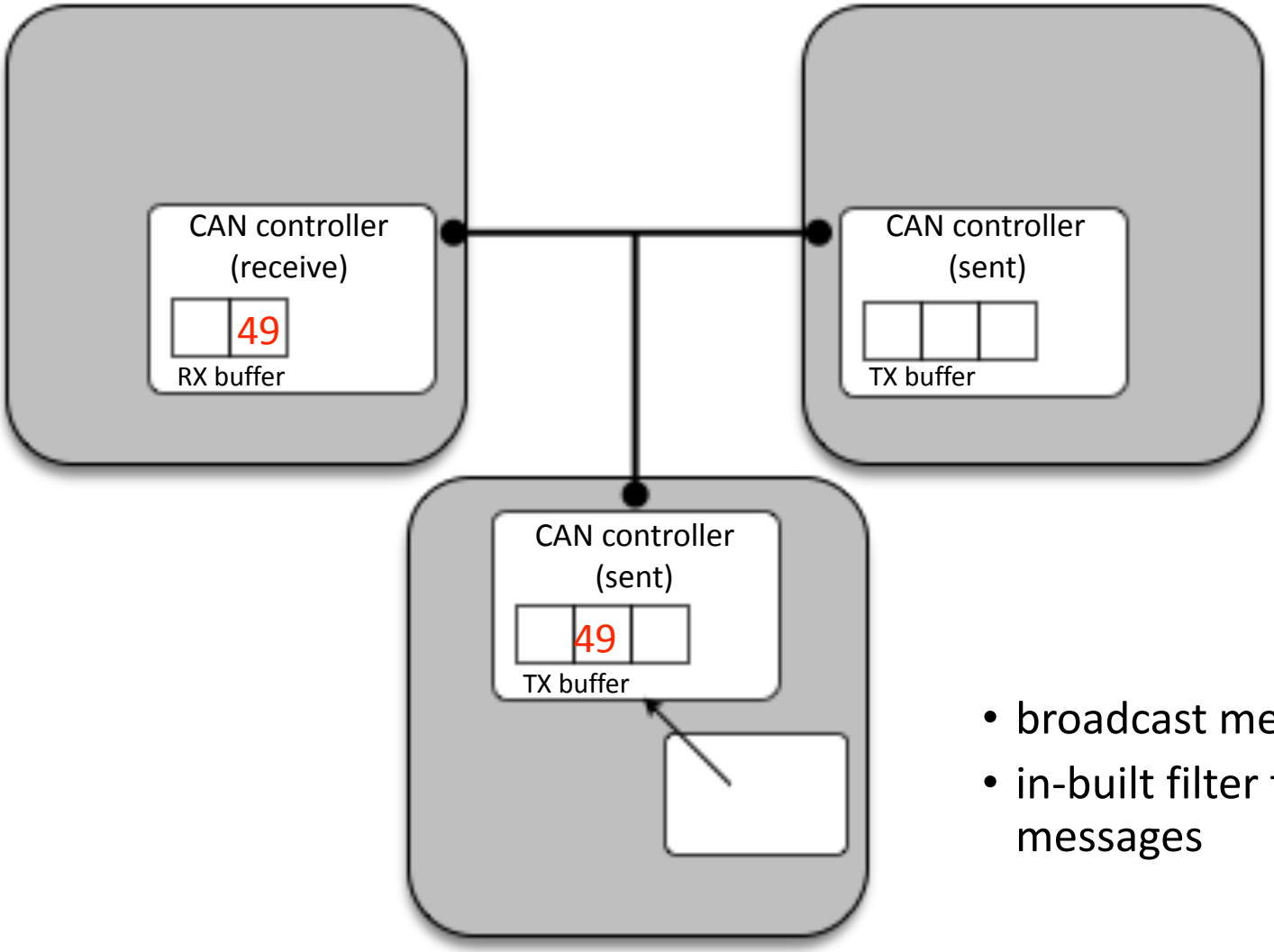
April 2017

www.data61.csiro.au

The Controller Area Network (CAN bus)

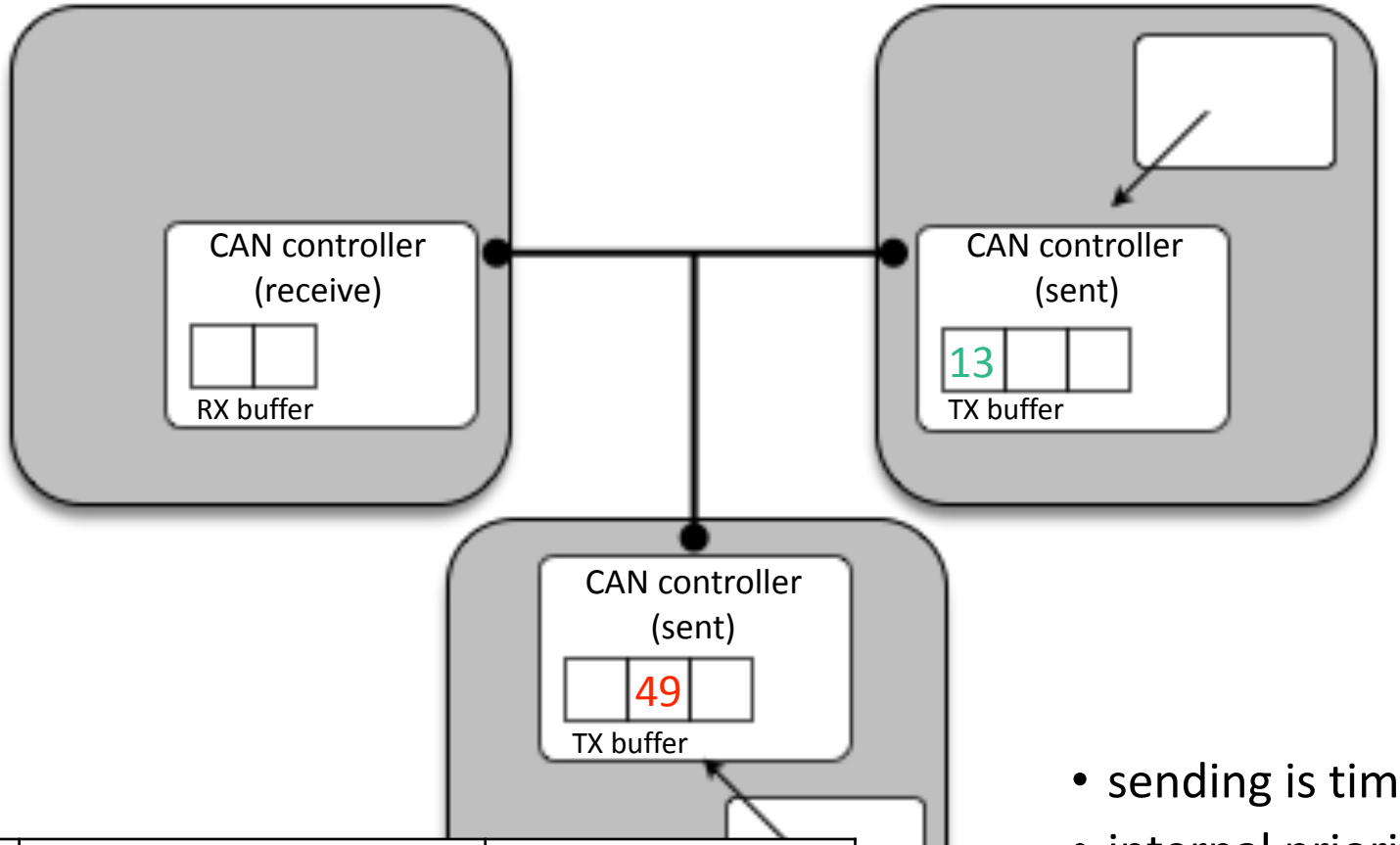
- Controller Area Network (CAN bus) is a vehicle bus standard
- Designed to allow microcontrollers and devices to communicate
- Robert Bosch GmbH (80s/1991)

CAN bus - Mechanism I



- broadcast mechanism
- in-built filter to ignore messages

CAN bus - Mechanism II



	CAN ID								Data
Comp 1	0	0	0	0	1	1	0	1	…
Comp 2	0	0	0	0	1	1	1		

- sending is timed
- internal prioritisation mechanism

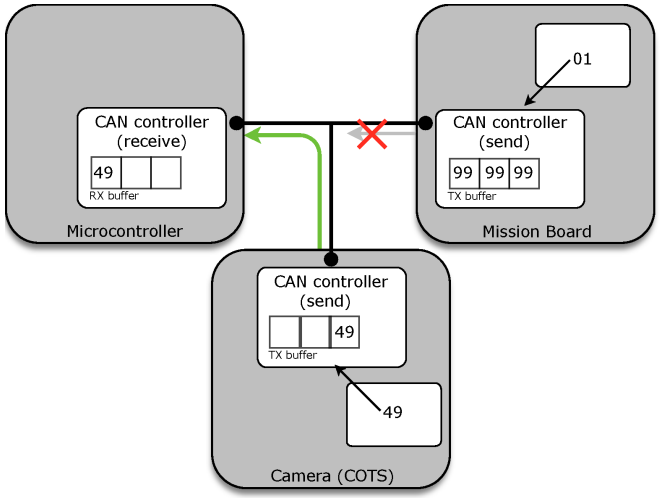
CAN bus - Limitations

- Designed for sensor data
  - payload only 8 bytes
  - no security/authentication
  - cannot send large messages

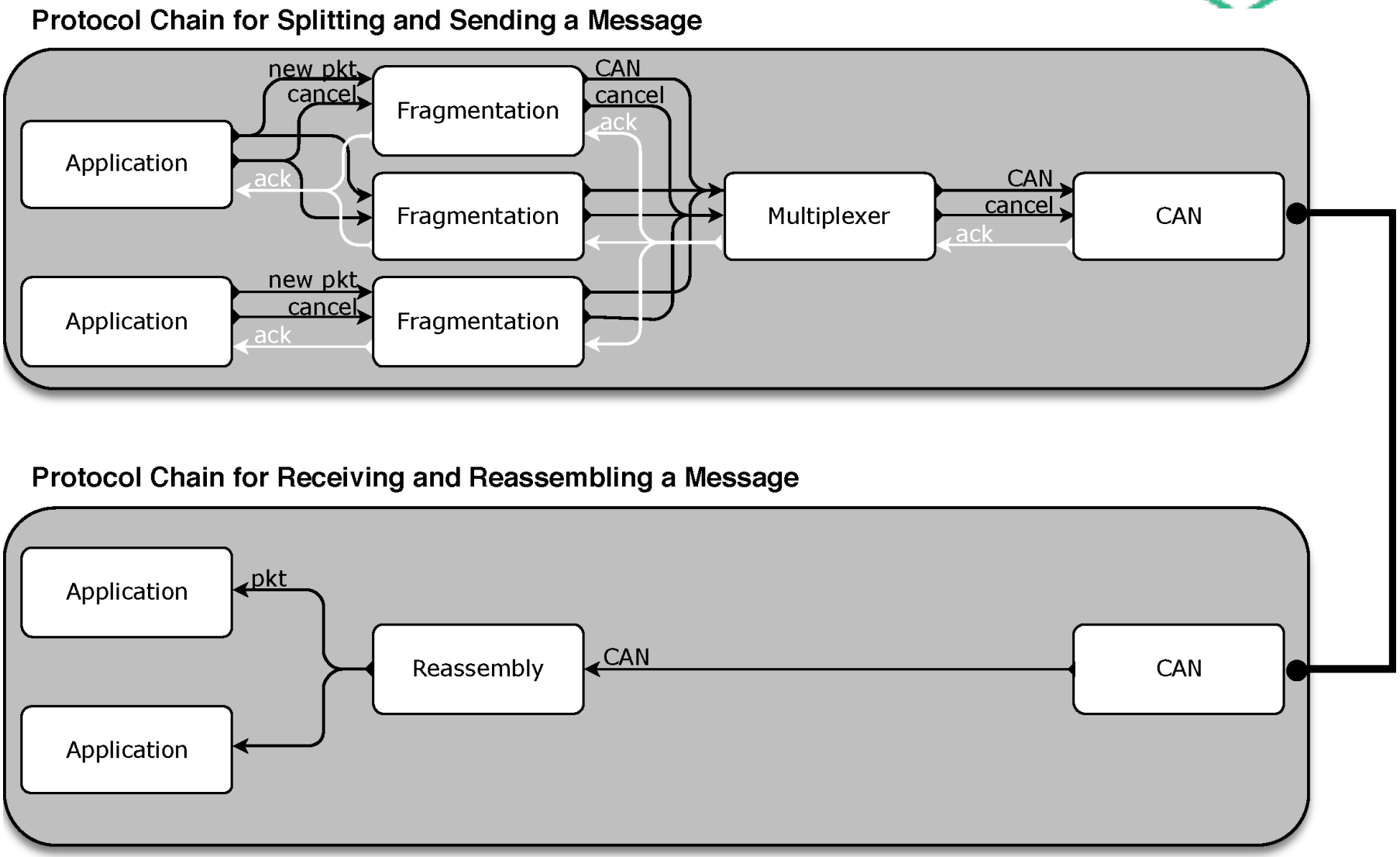
  Need for Fragmentation and Reassembly

- Priority Inversion
  - high-priority messages can be blocked
  - in-built prioritisation is not enough

  Need for Prioritisation



Split, Send, Reassemble

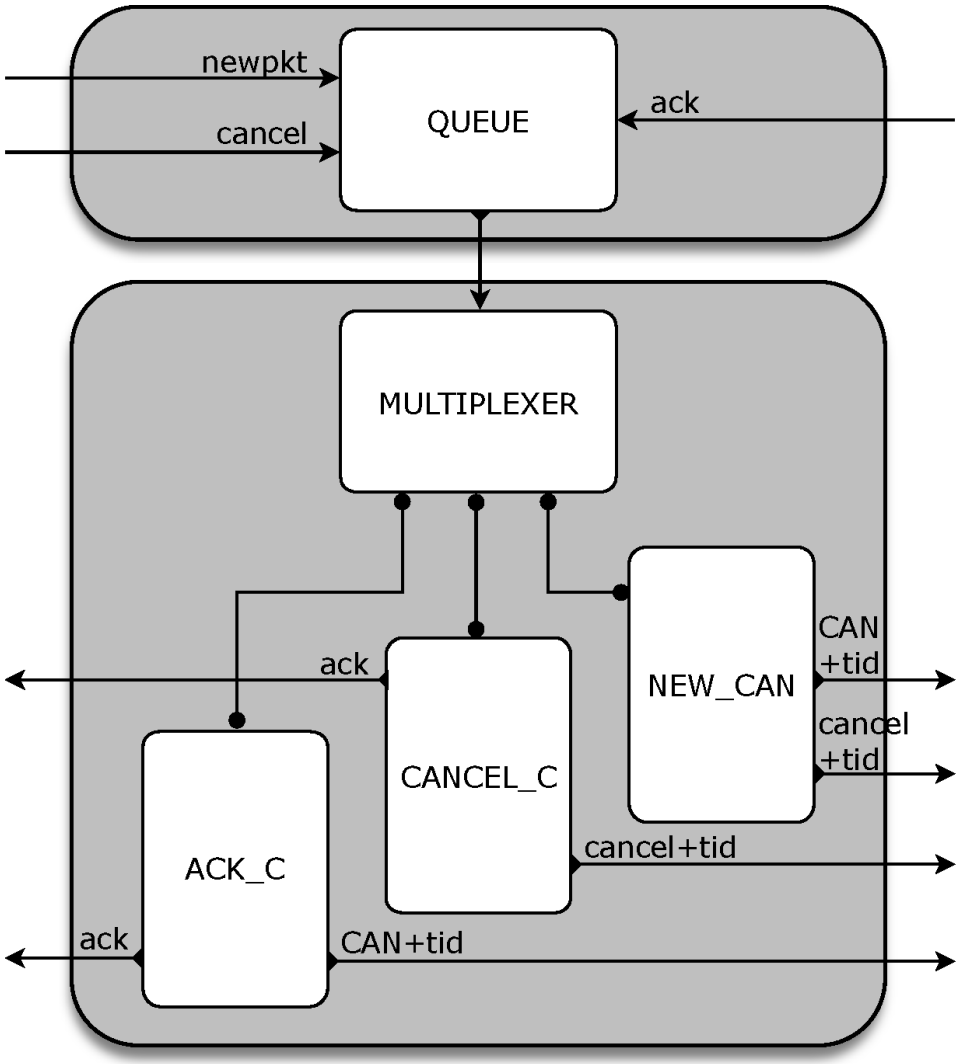


Formal Specification

- Developed formal (unambiguous) specification

- Used process algebra AWN
  - AWN was successfully used to model and verify AODV
    - offers data structure, and different sending mechanisms such as broadcast, unicast, etc.
    - AWN has a formally defined semantics
  - offers support for
    - model checking (Uppaal/mCRL2)
      (there is a sound translation, not yet fully implemented)
    - interactive theorem provers (Isabelle/HOL)
    - pen-and-paper reasoning

The Multiplexer



The Multiplexer - Main Loop

Process 9 Multiplexer—Main Loop

$\texttt{MULTIPLEXER}_H(\texttt{prio},\texttt{txs}) \stackrel{def}{=}$

1. **receive**(msg) .
2. (
3. [msg = can(cid,data)] /* new fragment */
4. NEW_CAN$_H$(msg,prio,txs)
5. + [msg = cancel(cid)] /* cancellation message received */
6. CANCEL_C$_H$(cid,prio,txs)
7. + [msg = msgd(tid,ack(suc))] /* message from CAN controller */
8. ACK_C$_H$(suc,tid,prio,txs)
9.)

Process Algebra AWN

- Description Language (Syntax)

$X(exp_1, \ldots, \exp_n)$	process calls
$P + Q$	nondeterministic
$[\varphi]P$	if-construct (guard)
$[\![\mathtt{var} := exp]\!]P$	assignment followed
$\mathbf{broadcast}(ms).P$	broadcast
$\mathbf{groupcast}(dests, ms).P$	groupcast
$\mathbf{unicast}(dest, ms).P \blacktriangleright Q$	unicast
$\mathbf{send}(ms).P$	send
$\mathbf{receive}(\mathtt{msg}).P$	receive
$\mathbf{deliver}(data).P$	deliver

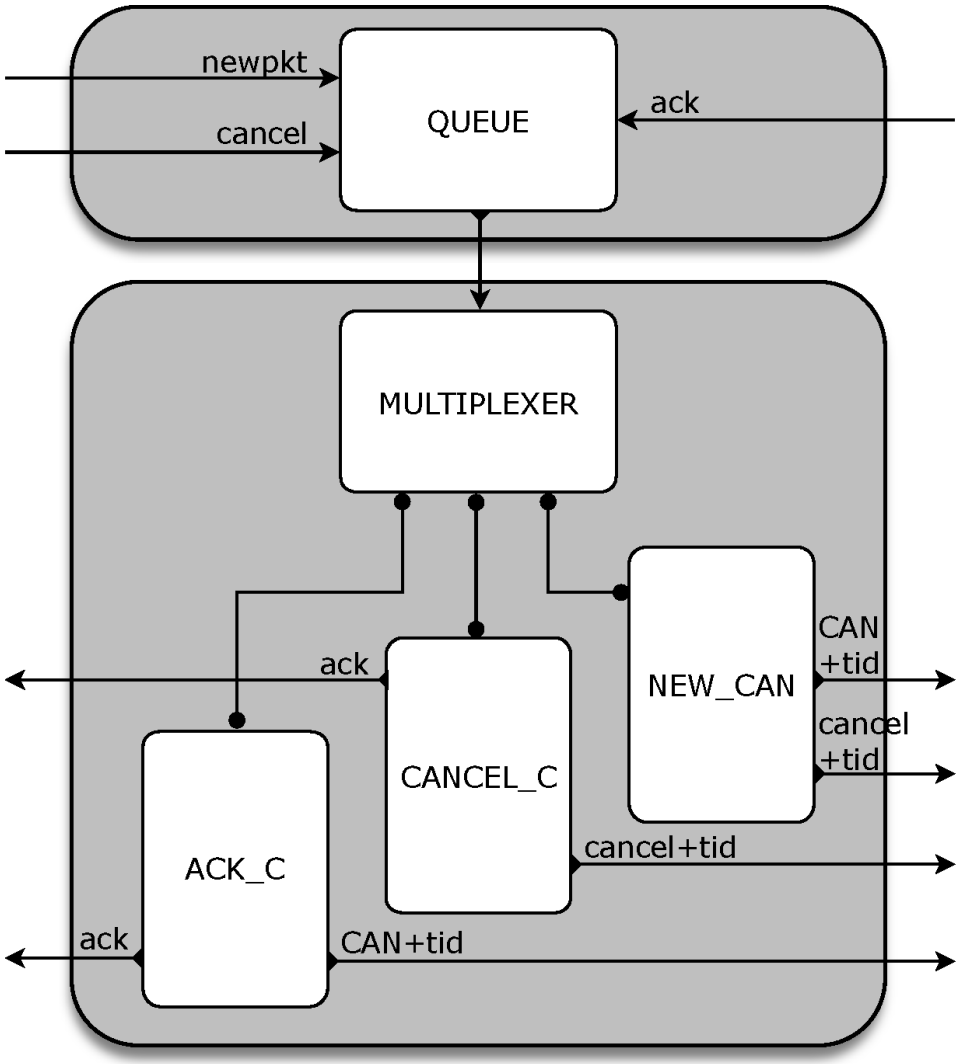
Developed Process Algebra

- Description Language (Syntax)

$[\varphi]P + [\neg\varphi]Q$	deterministic choice
$P(n) = [\![n := n+1]\!].P(n)$	loops

- Parallel Operator

The Multiplexer



Multiplexer - new CAN message

Process 10 New CAN Message Received

$\texttt{NEW\_CAN}_H(\texttt{msg},\texttt{prio},\texttt{txs}) \stackrel{def}{=}$

1. $[\ \texttt{msg} = \texttt{can}(\texttt{cid},\texttt{data})\ ]$ /* distill cid out of msg */
2. $[\![\texttt{prio}(\texttt{cid}) := \texttt{msg}]\!]$ /* store message in priority queue */
3. (
4. $[\ \texttt{cid} \in \texttt{n\_best}(\texttt{prio})\ ]$ /* message should be scheduled */
5. (
6. $[\ \texttt{tx}_{\texttt{cid}}(\texttt{tid},\texttt{txs}) = \bot_{\texttt{cid}}\ ]$ /* TX buffer tid is free */
7. $[\![\texttt{txs}(\texttt{tid}) := (\texttt{cid},\texttt{false})]\!]$
8. $\textbf{unicast}(\texttt{C}_H, \texttt{msgd}(\texttt{tid},\texttt{msg}))$. /* pass message to CAN driver, to put in free slot */
9. $\texttt{MULTIPLEXER}_H(\texttt{prio},\texttt{txs})$
10. $+\ [\ \forall \texttt{tid} \in \texttt{TX} : \texttt{tx}_{\texttt{cid}}(\texttt{tid},\texttt{txs}) \neq \bot_{\texttt{cid}}\ ]$ /* cancel message with lowest priority */
11. (
12. $[\![\texttt{wid} := \texttt{getWorstTX}(\texttt{txs})]\!]$ /* identify TX buffer containing lowest CAN ID */
13. (
14. $[\ \texttt{tx}_{\texttt{abort}}(\texttt{wid},\texttt{txs}) = \texttt{false}\ ]$ /* TX buffer wid is still active */
15. $[\![\texttt{txs}(\texttt{wid}) := (\texttt{tx}_{\texttt{cid}}(\texttt{wid},\texttt{txs}),\texttt{true})]\!]$ /* set the abort-flag of buffer wid */
16. $\textbf{unicast}(\texttt{C}_H, \texttt{msgd}(\texttt{wid},\texttt{cancel}()))$. /* cancel contents of buffer wid */
17. $\texttt{MULTIPLEXER}_H(\texttt{prio},\texttt{txs})$
18. $+\ [\ \texttt{tx}_{\texttt{abort}}(\texttt{wid},\texttt{txs}) = \texttt{true}\ ]$ /* TX was already asked to clean up */
19. $\texttt{MULTIPLEXER}_H(\texttt{prio},\texttt{txs})$
20.)
21.)
22.)
23. $+\ [\ \texttt{cid} \notin \texttt{n\_best}(\texttt{prio})\ ]$ /* message not important enough to be scheduled right now */
24. $\texttt{MULTIPLEXER}_H(\texttt{prio},\texttt{txs})$
25.)

Overall Protocol

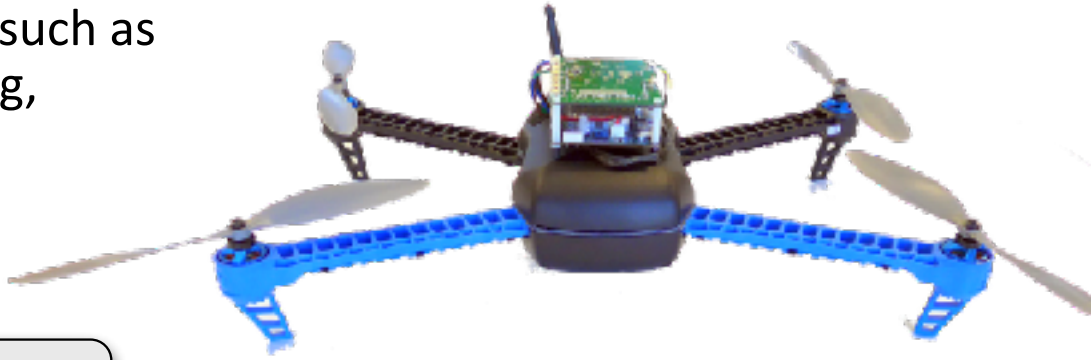
- Specification
  - 13 processes written in AWN
  - around 25 function
  - unambiguous

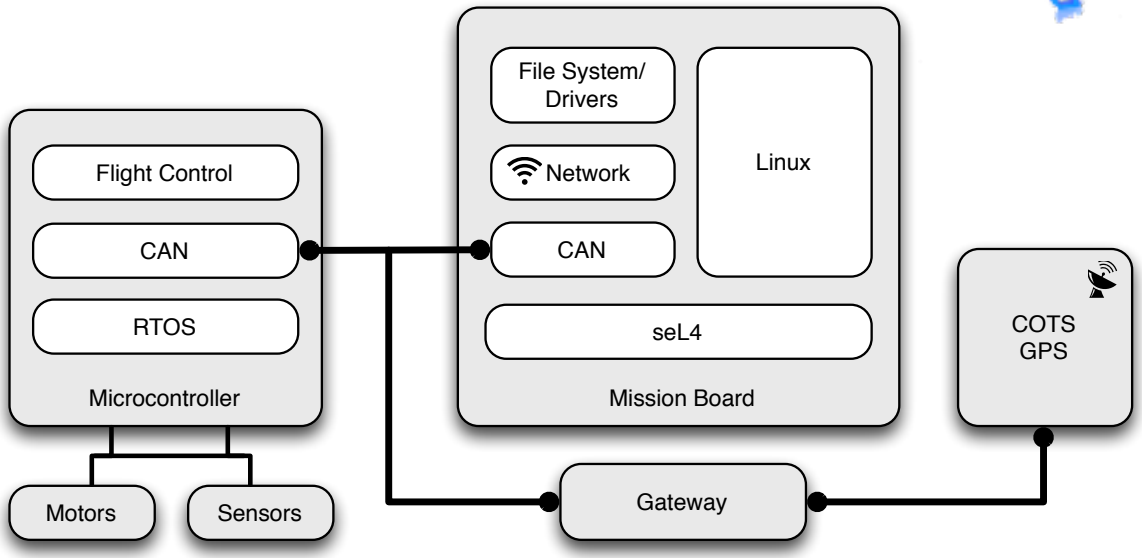
- Implementation
  - implemented by Galois Inc. (US)
  - in Ivory (Haskell)

- Features
  - can send messages of arbitrary length
  - solves problem of priority inversion

Research Vehicle

- Developed within DARPA's SMACCM program (Secure Mathematically-Assured Composition of Control Models)
- cooperation by various partners, such as Galois Inc., Rockwell Collins, Boing, U Minnesota





Desired Properties

- Unreachability of ERROR states (“undesired behaviour”)
- The protocol is deadlock free
- Each component is deadlock free
- Any message received has been sent
- Any message sent is received
  (under side conditions)
- Buffers have maximal (finite) length
- The application layer can always succeed to submit new messages

Assumptions

- Assumptions on the CAN bus
  - perfect channel: every message sent will be received by all nodes
  - possibility that a CAN message is sent and received twice (possibly because of CAN's error frame)
  - messages sent over the CAN bus are not reordered (CAN controller allows overtaking)

- Requirements on the Input Data
  - every message type (CAN ID) has a unique sender
  - every message split has a fixed length
  - distribution of the message IDs is fixed at compile time
  - application layer, after submission of a message, will wait for an acknowledgement (positive or negative) before submitting a new message
- Remark on CAN IDs: extended frame up to $2^{29} = 536{,}870{,}912$

Conclusion

- Specification
  - designed formal (unambiguous) fragmentation protocol *on top* of CAN bus
  - solved priority inversion problem of CAN

- Analysis
  - first analysis performed, using Uppaal (component-wise)
  - combination of components via Rely/Guarantee Reasoning (manual, not perfectly formal yet)

Conclusion

- Specification
  - designed formal (unambiguous) fragmentation protocol *on top* of CAN bus
  - solved priority inversion problem of CAN

- Analysis
  - first analysis performed, using Uppaal (component-wise)
  - combination of components via Rely/Guarantee Reasoning (manual, not perfectly formal yet)

- Additional Component
  - Gateway to avoid DoS and

Trustworthy Systems
Peter Höfner

t +61 2 9490 5861
e peter.hoefner@data61.csiro.au
w www.data61.csiro.au

www.data61.csiro.au

Related Work

- *ISO-TP or ISO 15765-2*
  - international standard
  - splits longer messages into multiple frames up to 4095 bytes
  - no extra prioritisation
- Shin follows the spirit of ISO-TP
  - capacity of 6 bytes only (more overhead)
- *TP 2.0 or VW TP 2.0*
  - sends a couple of messages to establish a “channel” between the sender and the recipients
  - then exchanges and sets up channel parameters such as the number of frames
  - overhead lies in the setup-phase.
- *CANopen*
  - fragmentation is a subprotocol
  - too much of an overhead.