Formalization of WAMP

Tobias Oberstein

January 1, 2023

Formalization of WAMP

In the following, we define the WAMP protocol, not by directly describing the valid behavior of WAMP clients and routers as is done in the WAMP specification, but indirectly by *describing all possible valid message exchanges observable on transports*. This approached is claimed to be equivalent to the behavioral description of WAMP in the specification, which defines WAMP in operational terms for clients and routers. However, the behavioral description is not written in a formal language, which is easily written and read, but comes with its own problems. The approach taken here aims to define WAMP by specifying a formal language for WAMP traffic.

A given set of WAMP peers communicate by passing messages on point-to-point transports between each other. Of all possible messages (raw text or binary serialized messages) E that peers could send to and receive from each other technically

$$E = \{\text{all messages, binary strings of finite length}\}$$

only a *subset of messages* $M \subset E$ is valid WAMP communication

$$M = \{m \in E : m \text{ is a valid WAMP message }\}$$

System Model

A distributed system is partial synchronous if there are bounds on transmission delay, clock drift and processing time and these bounds are unknown.

GST (global stabilization time)

https://dev.to/skoya76/communication-models-of-distributed-systems-26i2

distributed system partial synchronous byzantine faults formel modeling

APPROACH:

formal definition of observable message traces, not system behavior

the system behavior is defined in plain english

the message traces that might be observed on a correctly behaving system are formally defined

https://wamp-proto.org/ https://wamp-proto.org/wamp_latest_ietf.html

https://github.com/crossbario/autobahn-python https://wamp-proto.org/implementations.html

Z3 SMT solver

SMTLib (.smt2)

https://github.com/Z3Prover/z3

z3-solver

https://github.com/Z3Prover/z3/tree/master/examples/python

https://pypi.org/project/z3-solver/

Z3 Python language bindings

https://colab.research.google.com/github/philzook58/z3_tutorial/blob/master/Z3%20Tutorial.ipynb

SIMULINK/STATEFLOW (S/S) https://www.mathworks.com/products/stateflow.html

Constraint Solvers for the Working PL Researcher - Nadia Polikarpova at PLMW@ICFP https://www.youtube.com/watch?v=rTOqg-f2rNM

Programming Z3 Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger, Microsoft Research https://theory.stanford.edu/~nikolaj/programmingz3.html

Z3 - a Tutorial Leonardo de Moura, Nikolaj Bjørner, Microsoft Research https://www.cs.colostate.edu/~cs440/spring19/slides/z3-tutorial.pdf

START FROM: https://github.com/crossbario/autobahn-js/blob/master/packages/autobahn/test/test_pubsub_

Logical Physical Clocks Sandeep S. Kulkarni1 , Murat Demirbas2 , Deepak Madeppa2 , Bharadwaj Avva2 , and Marcelo Leone 1 1 Michigan State University, 2 University of Buffalo, SUNY

"HLC does not require waiting out the clock uncertainty, since it is able to record causality relations within this uncertainty interval using the HLC update rules"

"HLC can be implemented using 64 bits space, and is backwards compatible with NTP clocks. Moreover, HLC only reads NTP clock values but does not change it."

https://cse.buffalo.edu/~demirbas/publications/hlc.pdf https://arxiv.org/pdf/1707.07699.pdf

System structure

Peer behavior

Crash-stop (fail-stop): A peer is faulty if it crashes (at any moment). After crashing, it stops executing forever.

Crash-recovery (fail-recovery): A peer may crash at any moment, losing its in-memory state. It may resume executing sometime later.

Byzantine (fail-arbitrary): A peer is faulty if it deviates from the protocol. Faulty peers may do anything, including crashing or malicious behavior.

unintentional (buggy) intentional (malicious)

A peer that is not faulty is called "correct".

Transport behavior

WAMP transports are **message based** communication channels which are **bidirectional**, **reliable** and **ordered** (§1.3.1.).

In general, a model of a transport or communication channel can assume different behaviors:

1. **Reliable and Ordered**: A message is received at the far end of a transport if and only if it is sent at the near end of that transport. Messages are transported ordered, that is, a message 1 is received before message 2 at the far end of a transport if and only if message 1 is sent before message 2 at the near end of that transport.
2. **Reliable**: A message is received at the remote end of a transport if and only if it is sent at the near end of that transport. Messages may be reordered.
3. **Fair-loss**: Messages may be lost, duplicated or reordered in a transport. If you keep retrying, a message eventually gets throught.
4. **Arbitrary**: A third party or malicious adversary may interfere with messages and transports (eavesdrop, modify, drop, spoof, replay, inject).

Timing behavior

Synchronized: - Message latency (through a transport) fixed at known value - Global time at peers with skew no greater than a known upper bound

Synchronous: - Message latency (through a transport) no greater than a known upper bound - Local time at peers with constant clock and drift no greater than a known upper bound - Execution time at peers

Partially Synchronous

loosely synchronized clocks

Asynchronous

Message Exchange Language

A set of WAMP clients and routers communicating over WAMP transports are said to *correctly implement WAMP* if all message exchanges observed on transports are always valid.

Since transports are bidirectional, reliable and ordered, it is sufficient to specify only the messages as sent or received by the WAMP client end of a transport (§1.3.2.).

Let $MsgDirs$ be the set of message directions

$$MsgDirs = \{\text{Receive}, \text{Transmit}\}$$

denoting whether a WAMP message was sent from or received by the WAMP client (§1.4.3.) on a transport.

Let $MsgTypes$ be the set of message types

$$MsgTypes = \{\mathsf{HELLO}, \mathsf{WELCOME}, \mathsf{ABORT}, \mathsf{GOODBYE}, \dots\}$$

of WAMP messages sent or received by a client (§3.5.).

Then we can model a WAMP transport C_k as a function

$$C_k : \{1, 2, .., L(C_k)\} \longrightarrow MsgDirs \times MsgTypes$$

where $L(C_k)$ is the lifetime of transport C_k, with $C_k(1)$ being the first message sent or received at the WAMP Client over the transport after the transport has been connected, and $C_k(L(C_k))$ being the last message sent or received at the WAMP Client over the transport before the transport has been disconnected.

Constraints

Per publisher ordering guarantees

$$
\begin{aligned}
&\quad(\exists C_i \text{ and } i_1 < i_2 < i_3 < i_4 :\\
&\quad (C_i, i_1, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}) \in M \wedge\\
&\quad (C_i, i_2, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}) \in M \wedge\\
&\quad (C_i, i_3, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Publisher} : \texttt{session2}, \mathit{Payload} : \texttt{data1}) \in M \wedge\\
&\quad (C_i, i_4, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Publisher} : \texttt{session2}, \mathit{Payload} : \texttt{data2}) \in M)\\
&\Longrightarrow\\
&\quad(\exists C_j \text{ and } j_1 < j_2 < j_3 :\\
&\quad (C_j, j_1, \text{Receive}, \mathsf{WELCOME}, \mathit{Session} : \texttt{session2}) \in M \wedge\\
&\quad (C_j, j_2, \text{Transmit}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}) \in M \wedge\\
&\quad (C_j, j_3, \text{Transmit}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}) \in M)
\end{aligned} \tag{1}
$$

Multiple publishers

$$
\begin{aligned}
&\quad(\exists C_i \text{ and } i_1 < i_2 < i_3 < i_4 :\\
&\quad (C_i, i_1, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}) \in M \wedge\\
&\quad (C_i, i_2, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}) \in M \wedge\\
&\quad (C_i, i_3, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Publisher} : \texttt{session2}, \mathit{Payload} : \texttt{data1}) \in M \wedge\\
&\quad (C_i, i_4, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Publisher} : \texttt{session3}, \mathit{Payload} : \texttt{data2}) \in M)\\
&\Longrightarrow\\
&\quad(\exists C_j \text{ and } j_1 < j_2 :\\
&\quad (C_j, j_1, \text{Receive}, \mathsf{WELCOME}, \mathit{Session} : \texttt{session2}) \in M \wedge\\
&\quad (C_j, j_2, \text{Transmit}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}) \in M)\\
&\quad\wedge\\
&\quad(\exists C_k \text{ and } k_1 < k_2 :\\
&\quad (C_k, k_1, \text{Receive}, \mathsf{WELCOME}, \mathit{Session} : \texttt{session3}) \in M \wedge\\
&\quad (C_k, k_2, \text{Transmit}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}) \in M)
\end{aligned} \tag{2}
$$

Examples

Big Silence

A system with transports that never sent or received any message:

$$M = \{\} \tag{3}$$

Single session joining and leaving

A system with a single transport C_1 which joins a realm (without authentication) and leaves again immediately:

$$\begin{aligned} M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\ &(C_1, 2, \text{Receive}, \mathsf{WELCOME}),\\ &(C_1, 3, \text{Transmit}, \mathsf{GOODBYE}),\\ &(C_1, 4, \text{Receive}, \mathsf{GOODBYE})\} \end{aligned} \tag{4}$$

Single session that is rejected

A system with a single transport C_1 which tries to join and is rejected:

$$\begin{aligned} M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\ &(C_1, 2, \text{Receive}, \mathsf{ABORT})\} \end{aligned} \tag{5}$$

Two session, one publisher and one subscriber

A system with two transports C_1 and C_2, one publishing session, and one subscribing session receiving two events:

Valid 1 This is correct operation with the subscriber subscribed before the publisher publishes first event:

$$\begin{aligned} M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\ &(C_1, 2, \text{Receive}, \mathsf{WELCOME}),\\ &(C_1, 3, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\ &(C_1, 4, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\ &(C_1, 5, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data1}),\\ &(C_1, 6, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data2}),\\ &(C_1, 7, \text{Transmit}, \mathsf{GOODBYE}),\\ &(C_1, 8, \text{Receive}, \mathsf{GOODBYE})\}\\ &\cup\\ &\{(C_2, 1, \text{Transmit}, \mathsf{HELLO}),\\ &(C_2, 2, \text{Receive}, \mathsf{WELCOME}),\\ &(C_2, 3, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}),\\ &(C_2, 4, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}),\\ &(C_2, 5, \text{Transmit}, \mathsf{GOODBYE}),\\ &(C_2, 6, \text{Receive}, \mathsf{GOODBYE})\\ &\} \end{aligned} \tag{6}$$

Valid 2 This is correct operation, but the subscriber did not receive any of the two events because of one of these reasons:

- `session1` left before `session2` joined
- `session1` joined after `session2` left
- `session1` :: $\mathsf{PUBLISH}$ was unacknowledged, and lost or dropped by the router

$$\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 7, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \mathsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_2, 3, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}),\\
&(C_2, 4, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}),\\
&(C_2, 5, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \mathsf{GOODBYE})\\
\}&
\end{aligned} \tag{7}$$

Valid 3 This is correct operation, but the subscriber did only receive one of the two events because of one of these reasons:

- `session1` :: $\mathsf{SUBSCRIBE}$ and `session1` :: $\mathsf{PUBLISH}$ are in a race to the Broker.
- `session2` :: $\mathsf{PUBLISH}$ was unacknowledged, and lost or dropped by the router

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \textsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \textsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 6, \text{Receive}, \textsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data2}),\\
&(C_1, 7, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \textsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_2, 3, \text{Receive}, \textsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}),\\
&(C_2, 4, \text{Receive}, \textsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}),\\
&(C_2, 5, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \textsf{GOODBYE})\\
\}&
\end{aligned}
\tag{8}
$$

Valid 4 This is correct operation, but the subscriber did only receive one of the two events because of one of these reasons:

- `session1` :: GOODBYE and `session2` :: PUBLISH are in a race to the Broker.
- `session2` :: PUBLISH was unacknowledged, and lost or dropped by the router

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \mathsf{WELCOME}, \mathit{Session} : \texttt{session1}),\\
&(C_1, 3, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 6, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Publisher} : \texttt{session2}, \mathit{Payload} : \texttt{data1}),\\
&(C_1, 7, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \mathsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \mathsf{WELCOME}, \mathit{Session} : \texttt{session2}),\\
&(C_2, 3, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data1}),\\
&(C_2, 4, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}),\\
&(C_2, 5, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \mathsf{GOODBYE})\\
\}&
\end{aligned} \tag{9}
$$

Invalid 1 This is incorrect operation with an event received at a subscriber that was never published by any publisher:

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 5, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data1}),\\
&(C_1, 7, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \mathsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_2, 5, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \mathsf{GOODBYE})\\
\}&
\end{aligned} \tag{10}
$$

Invalid 2 This is incorrect operation with an event received at a subscriber that was never published by a publisher to the respective topic:

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \textsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \textsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 5, \text{Receive}, \textsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data1}),\\
&(C_1, 7, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \textsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_2, 3, \text{Receive}, \textsf{PUBLISH}, \mathit{Topic} : \texttt{uri2}, \mathit{Payload} : \texttt{data1}),\\
&(C_2, 5, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \textsf{GOODBYE})\\
\}&
\end{aligned}
\tag{11}
$$

Invalid 3 This is incorrect operation with an event received at a subscriber that was never published by a publisher with the respective payload:

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \textsf{SUBSCRIBE}, \mathit{Topic} : \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \textsf{SUBSCRIBED}, \mathit{Subscription} : \texttt{sub1}),\\
&(C_1, 5, \text{Receive}, \textsf{EVENT}, \mathit{Subscription} : \texttt{sub1}, \mathit{Payload} : \texttt{data1}),\\
&(C_1, 7, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \textsf{GOODBYE})\}\\
\cup&\\
\{&(C_2, 1, \text{Transmit}, \textsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \textsf{WELCOME}),\\
&(C_2, 3, \text{Receive}, \textsf{PUBLISH}, \mathit{Topic} : \texttt{uri1}, \mathit{Payload} : \texttt{data2}),\\
&(C_2, 5, \text{Transmit}, \textsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \textsf{GOODBYE})\\
\}&
\end{aligned}
\tag{12}
$$

Invalid 4 This is incorrect operation with events published by one publisher received in the wrong order at the one subscriber (this violates §7.1. Ordering Guarantees):

$$
\begin{aligned}
M = \{&(C_1, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_1, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_1, 3, \text{Transmit}, \mathsf{SUBSCRIBE}, \mathit{Topic}: \texttt{uri1}),\\
&(C_1, 4, \text{Receive}, \mathsf{SUBSCRIBED}, \mathit{Subscription}: \texttt{sub1}),\\
&(C_1, 5, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription}: \texttt{sub1}, \mathit{Payload}: \texttt{data2}),\\
&(C_1, 6, \text{Receive}, \mathsf{EVENT}, \mathit{Subscription}: \texttt{sub1}, \mathit{Payload}: \texttt{data1}),\\
&(C_1, 7, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_1, 8, \text{Receive}, \mathsf{GOODBYE})\}\\
\cup\\
\{&(C_2, 1, \text{Transmit}, \mathsf{HELLO}),\\
&(C_2, 2, \text{Receive}, \mathsf{WELCOME}),\\
&(C_2, 3, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic}: \texttt{uri1}, \mathit{Payload}: \texttt{data1}),\\
&(C_2, 4, \text{Receive}, \mathsf{PUBLISH}, \mathit{Topic}: \texttt{uri1}, \mathit{Payload}: \texttt{data2}),\\
&(C_2, 5, \text{Transmit}, \mathsf{GOODBYE}),\\
&(C_2, 6, \text{Receive}, \mathsf{GOODBYE})\\
\}&
\end{aligned}
\tag{13}
$$

References

- Distributed Systems 2.3: System models, Martin Kleppmann