Embedded MQTT Client in C++14

MQTT Client

ciere consulting

Michael Caisse

mcaisse@ciere.com | follow @MichaelCaisse

ciere.com

Part I

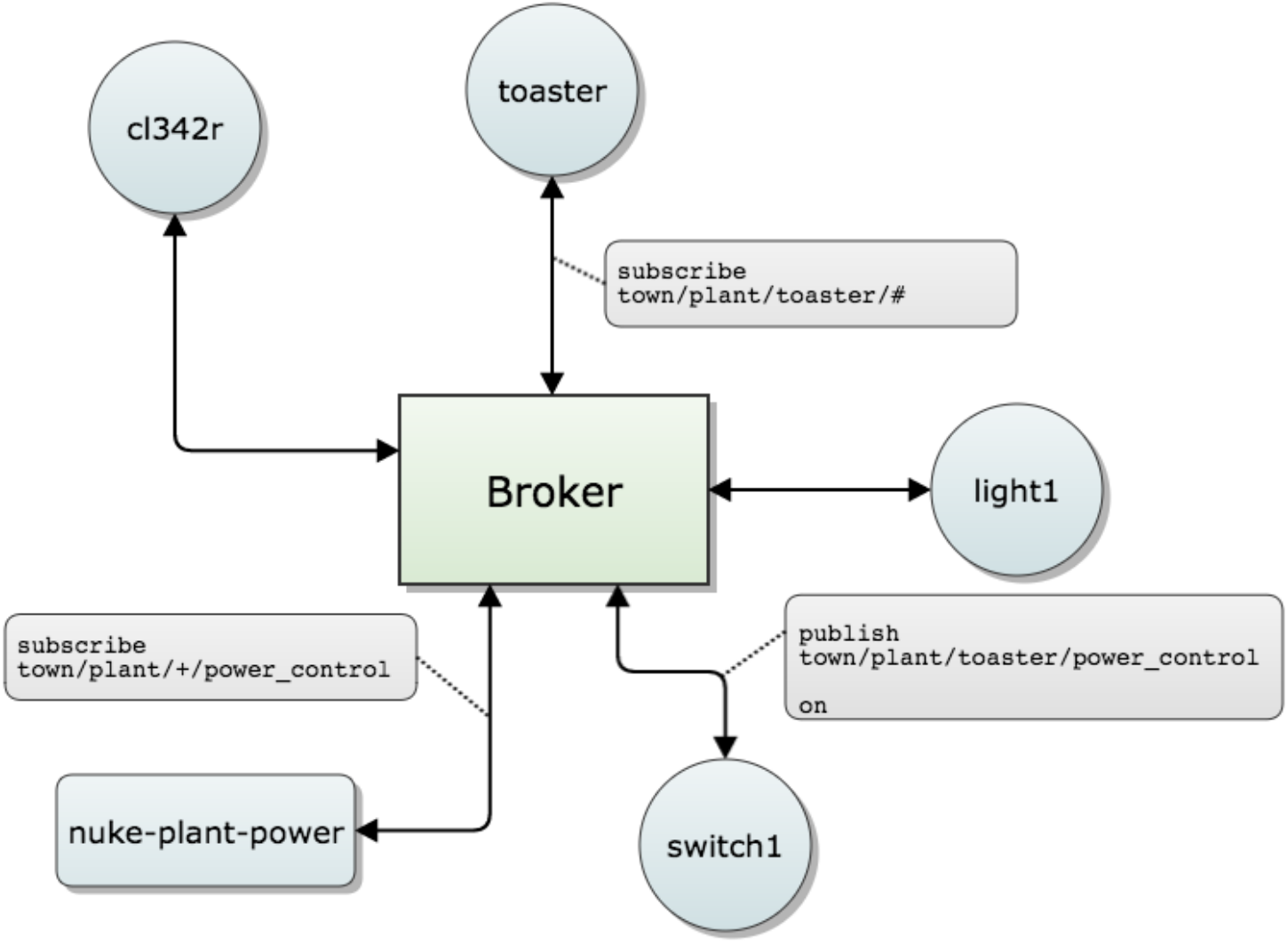
Introduction

Message Queue Telemetry Transport

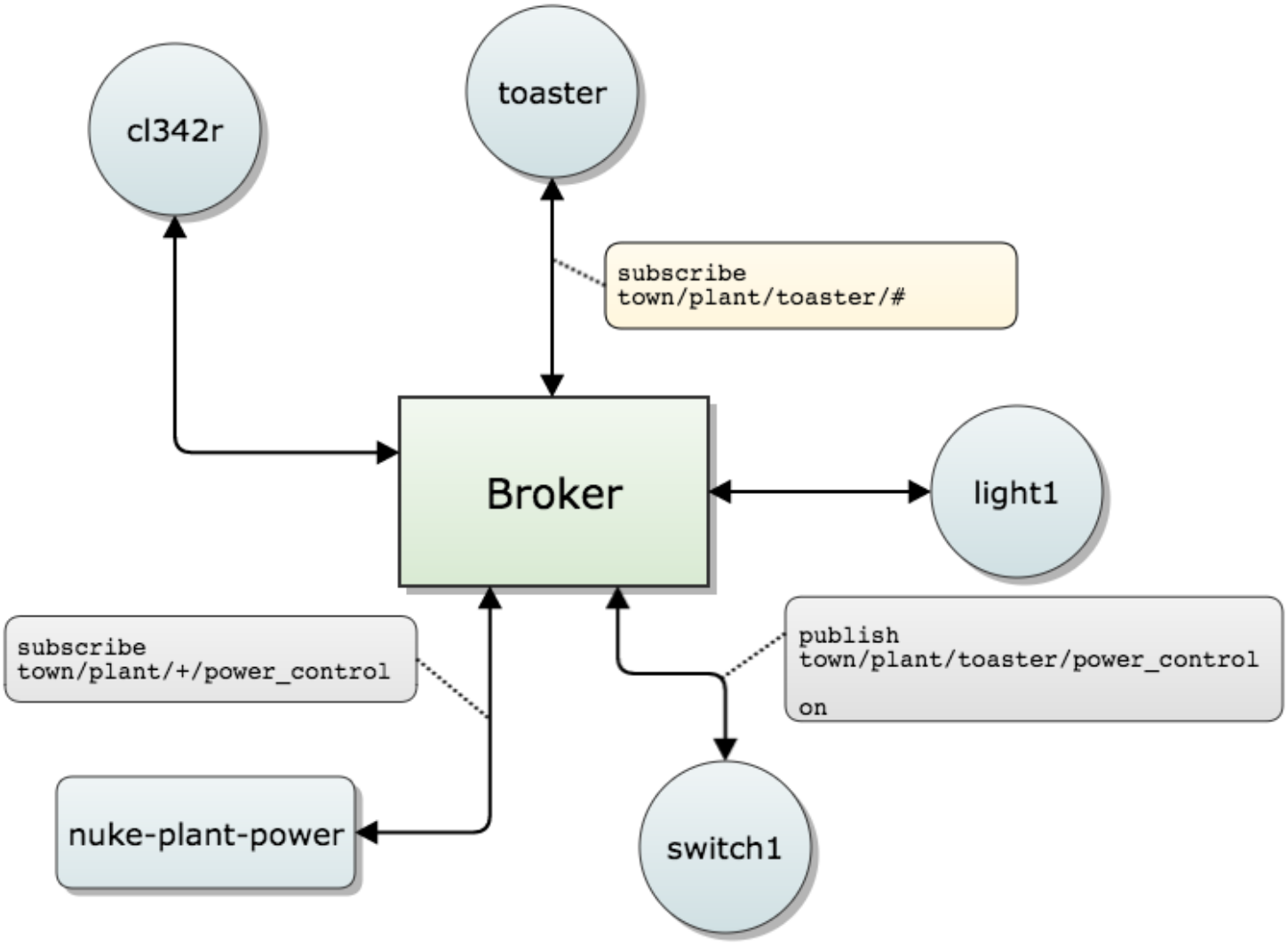
MQTT

- ISO Standard protocol
- Publisher/Subscriber model
- Considered light-weight
- Requires a reliable connection

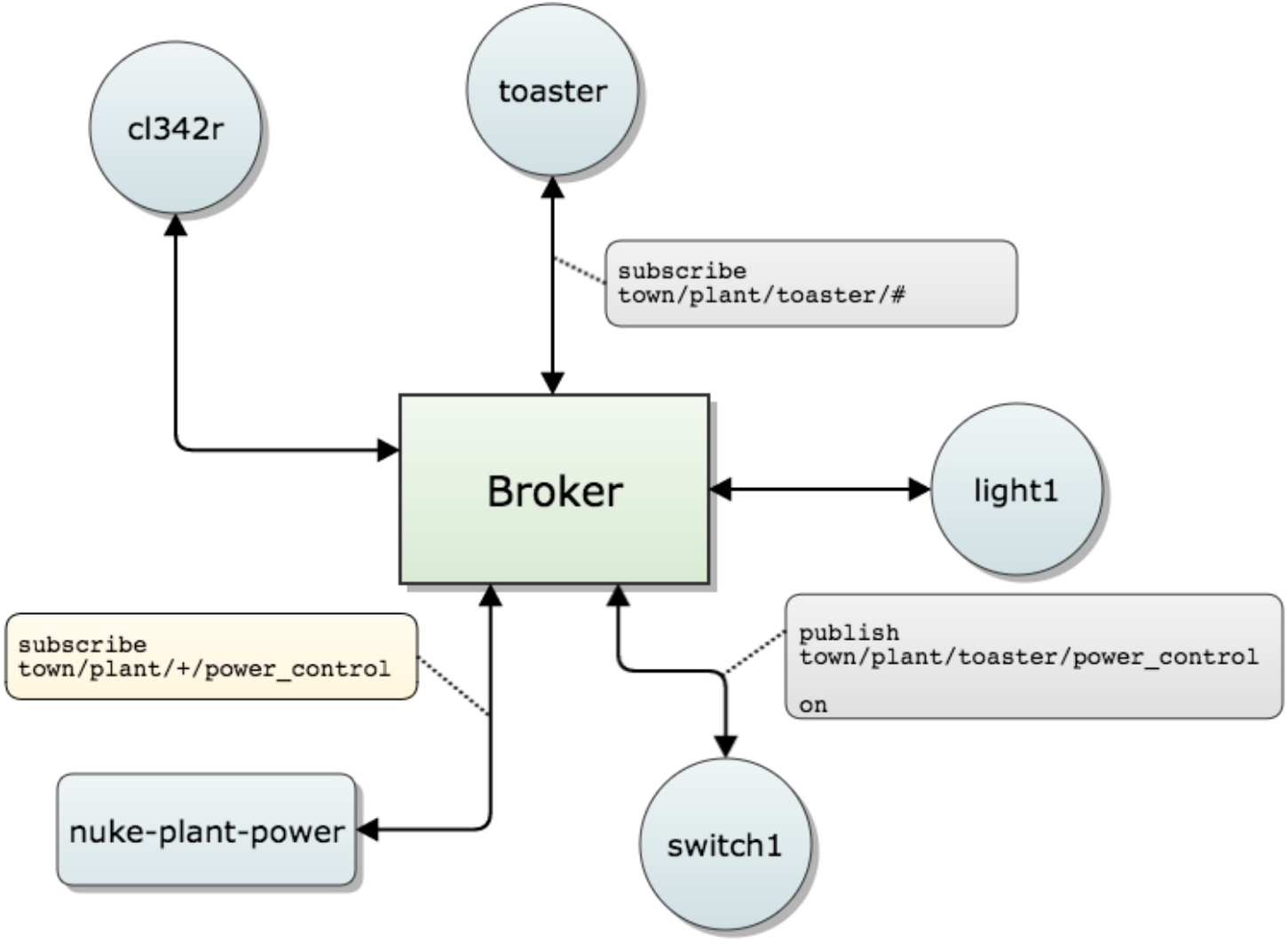
MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

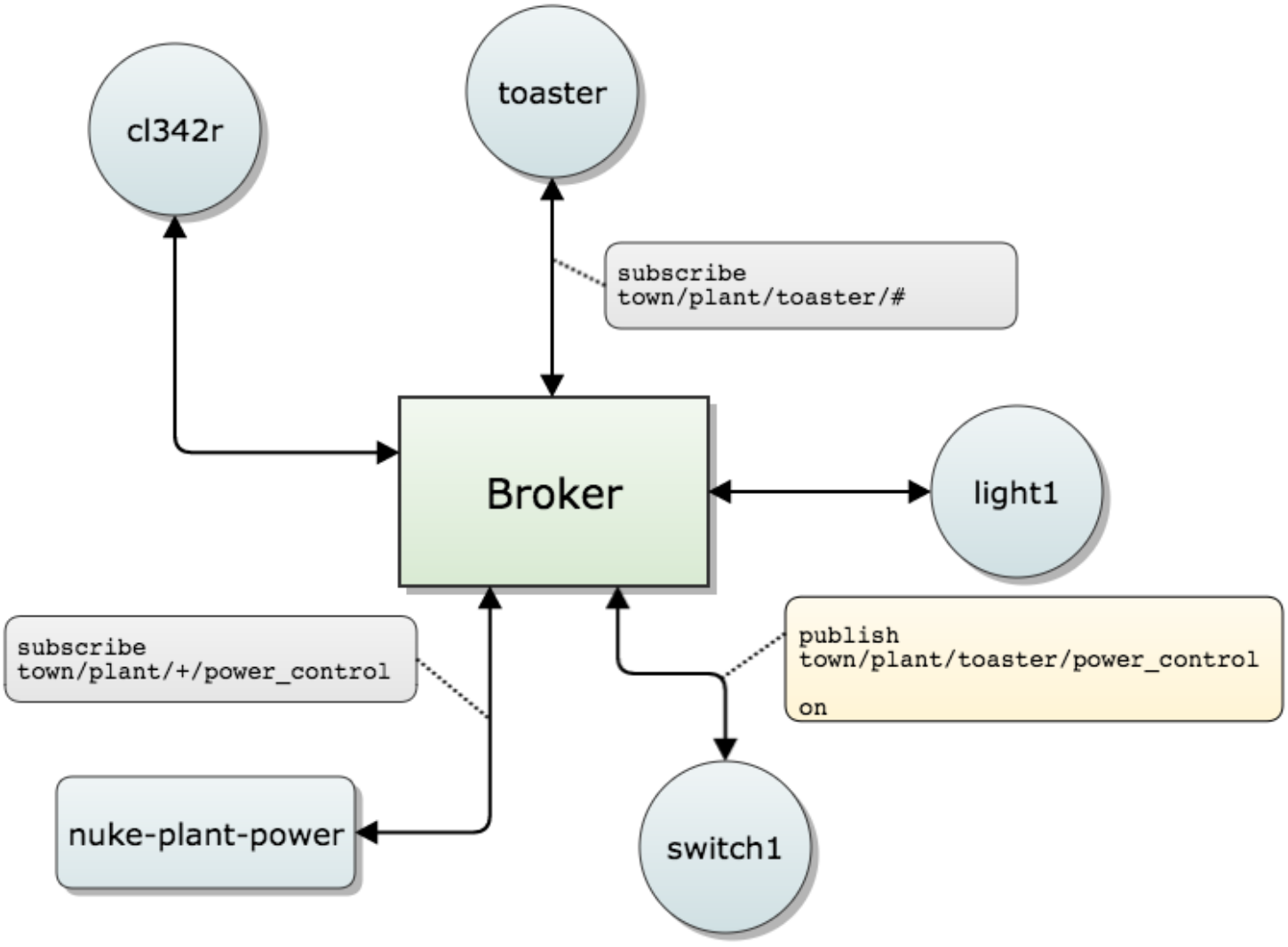
MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

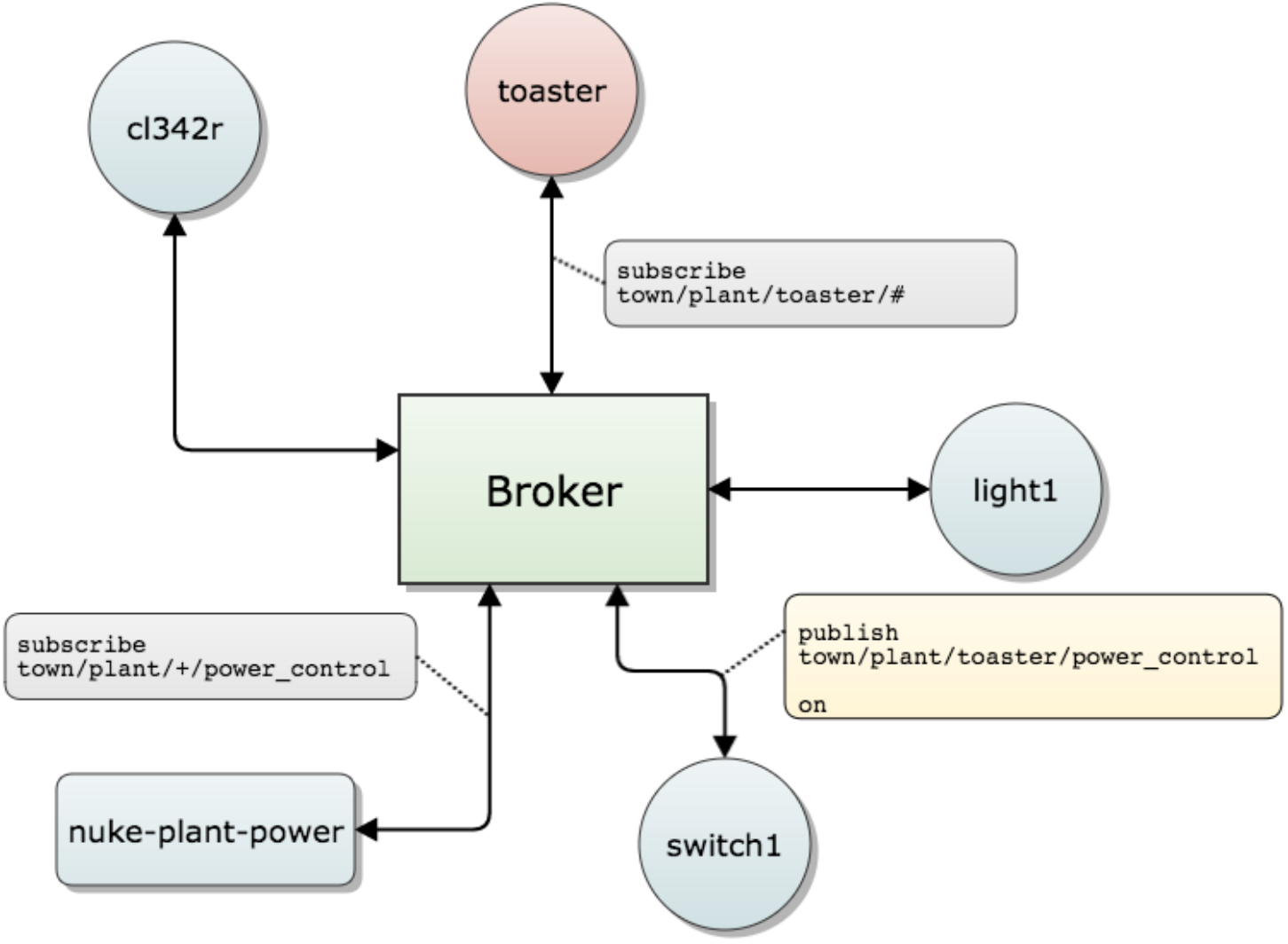
MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

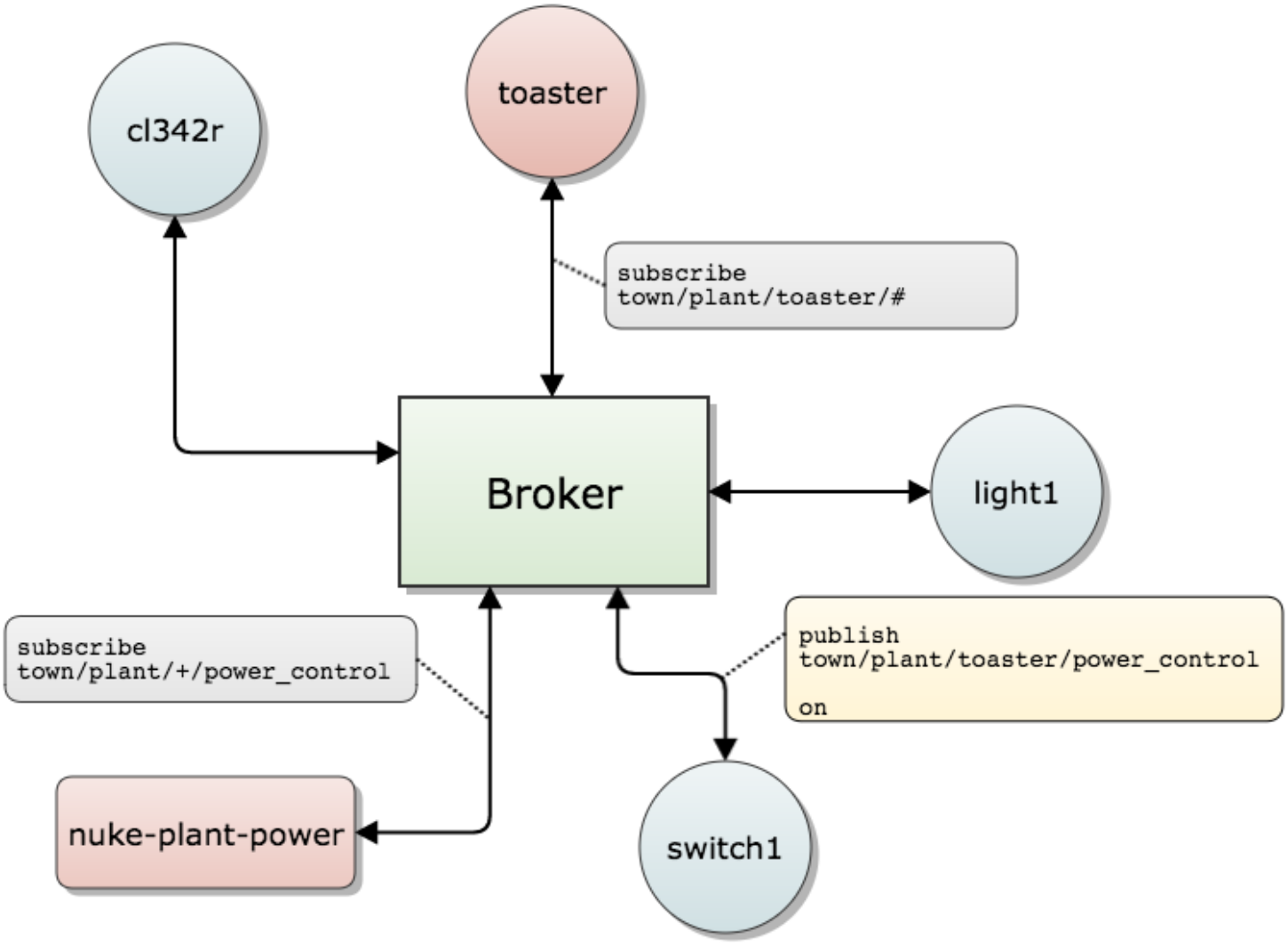
MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

MQTT


cl342r
toaster
subscribe
town/plant/toaster/#
Broker
light1
subscribe
town/plant/+/power_control
publish
town/plant/toaster/power_control
on
nuke-plant-power
switch1

MQTT

The basic messages

- connect
- disconnect
- subscribe
- unsubscribe
- publish
- ping

MQTT

Three levels of QoS (Quality of Service)

- At most once delivery - QoS 0
- At least once delivery - QoS 1
- Exactly once delivery - QoS 2

Where used?

MQTT is being used in large and small systems

- RabbitMQ to small devices
- Amazon IoT platform
- Lightbulbs, toasters, coffee makers... (IoT)
- medical equipment
- sensors
- phone applications

Embedded?

What is embedded?

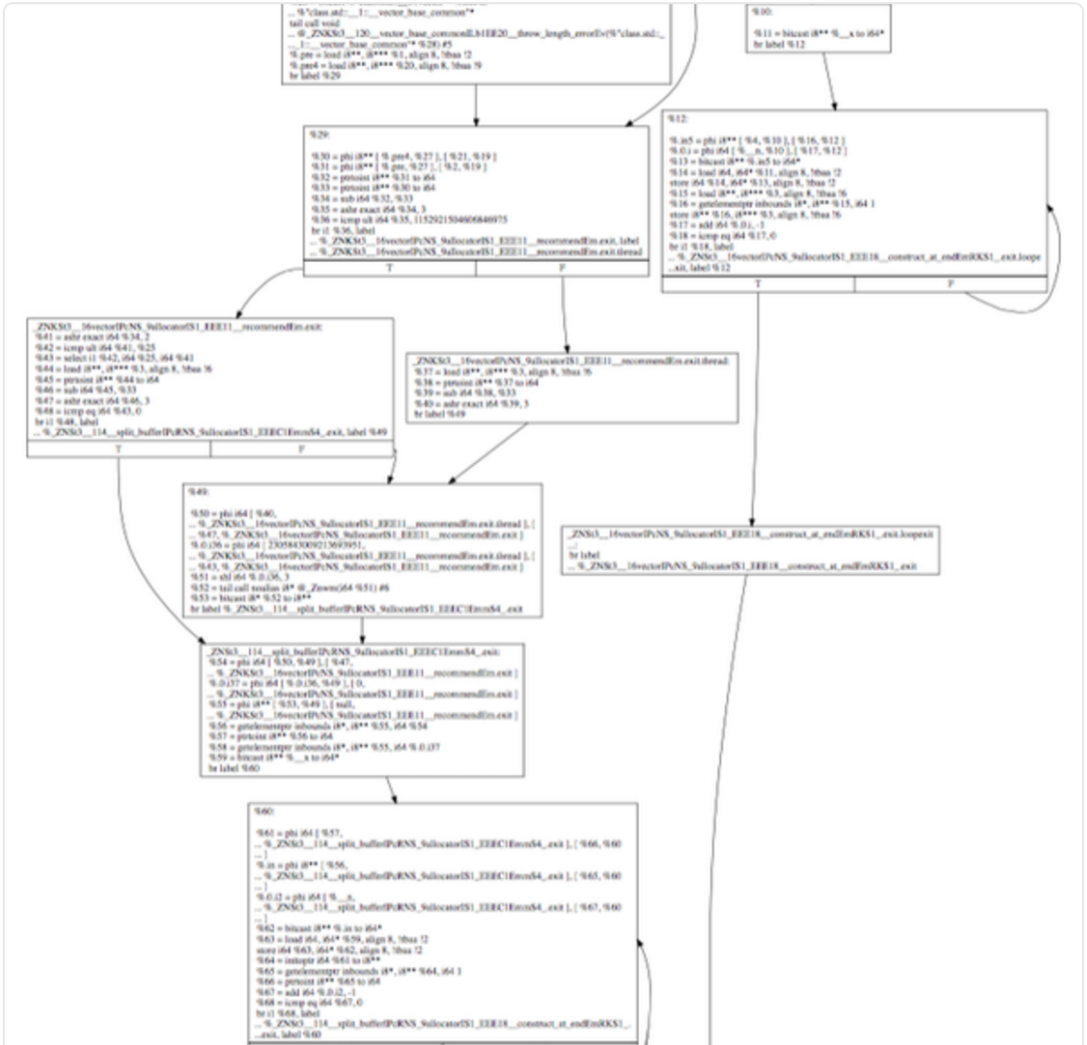
C++ on Embedded?

Wut!

Fiora Aeterna
@FioraAeterna

Follow

this is the code to set a vector to a constant size, initialized to zero.



#whatdoyouthink

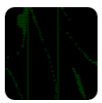
Michael Caisse
@MichaelCaisse

Top 4 reasons you have (or have heard) for not using C++ in "embedded" targets -- Ready? Go!

RETWEETS 2 LIKE 1

10:36 PM - 3 May 2016

#codebloat



Johnny T
@FrogShadeGarden

Follow

@MichaelCaisse More complicated regulatory approval. Smaller developer pool. Bloated binaries. OK, that's all I recall right now.

10:46 PM - 3 May 2016

#dontcrashmyrover

clement cole
@clement_cole

Follow

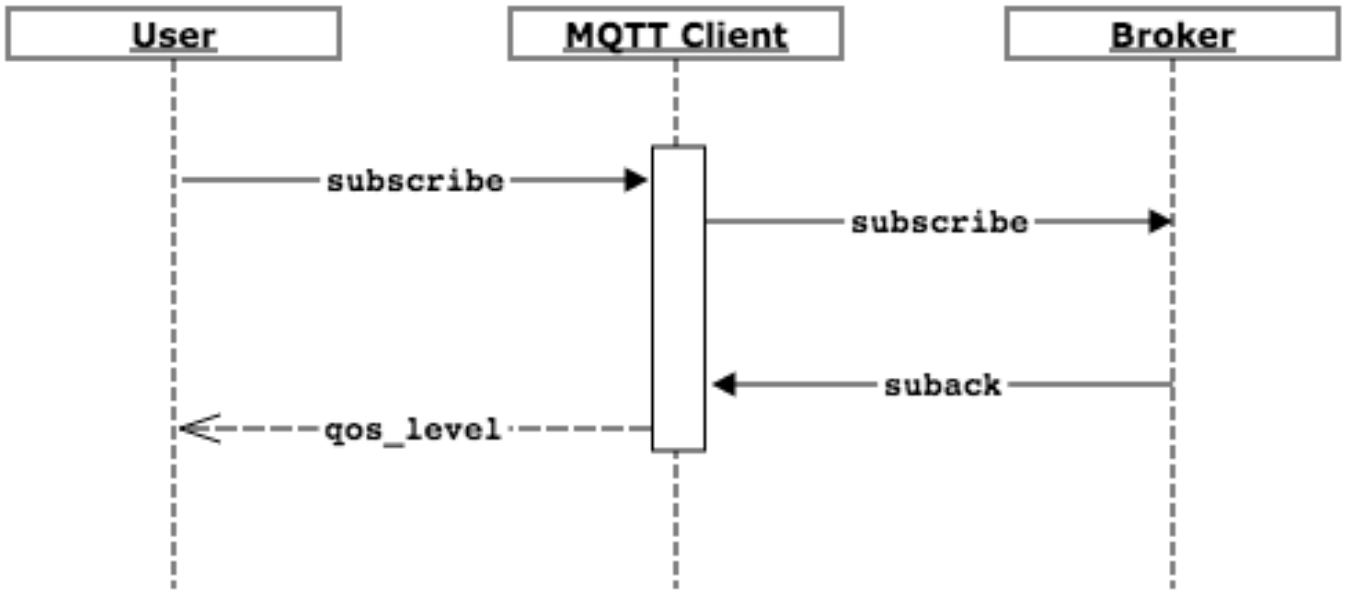
@MichaelCaisse CS programmers going wild by creating too many levels of abstractions ie templates and causing Mars Climate Observer to crash

6:36 PM - 11 May 2016

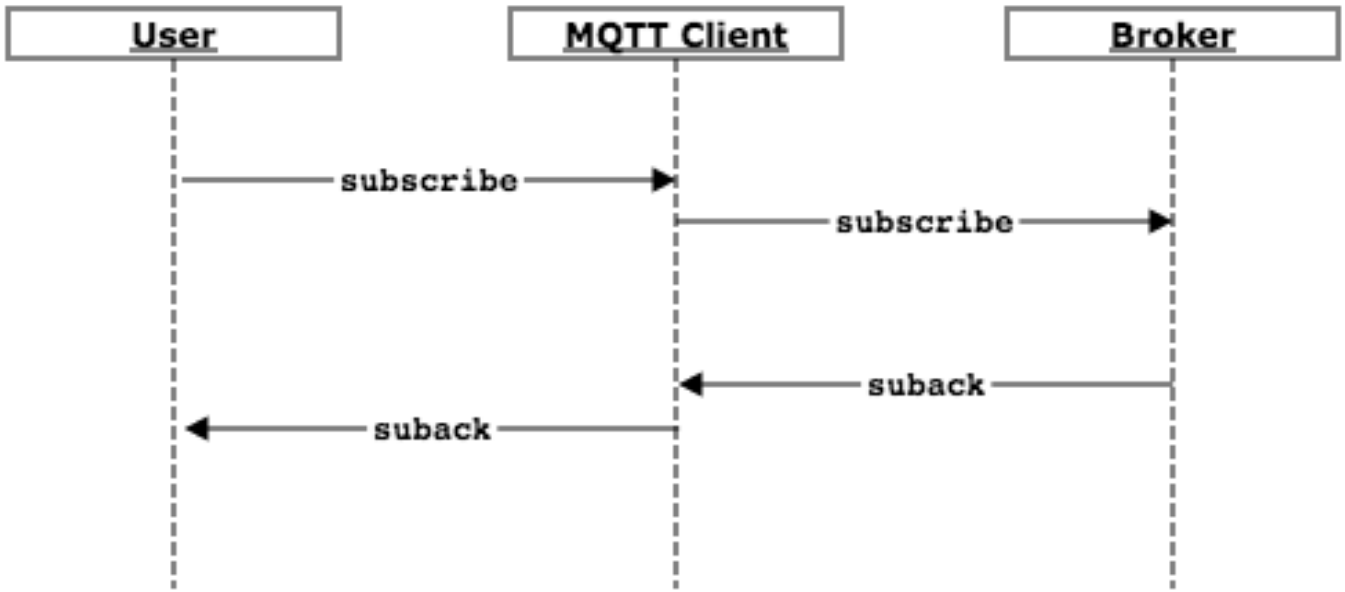
General Public Advice

Don’t do it!

Why another library?



Why another library?

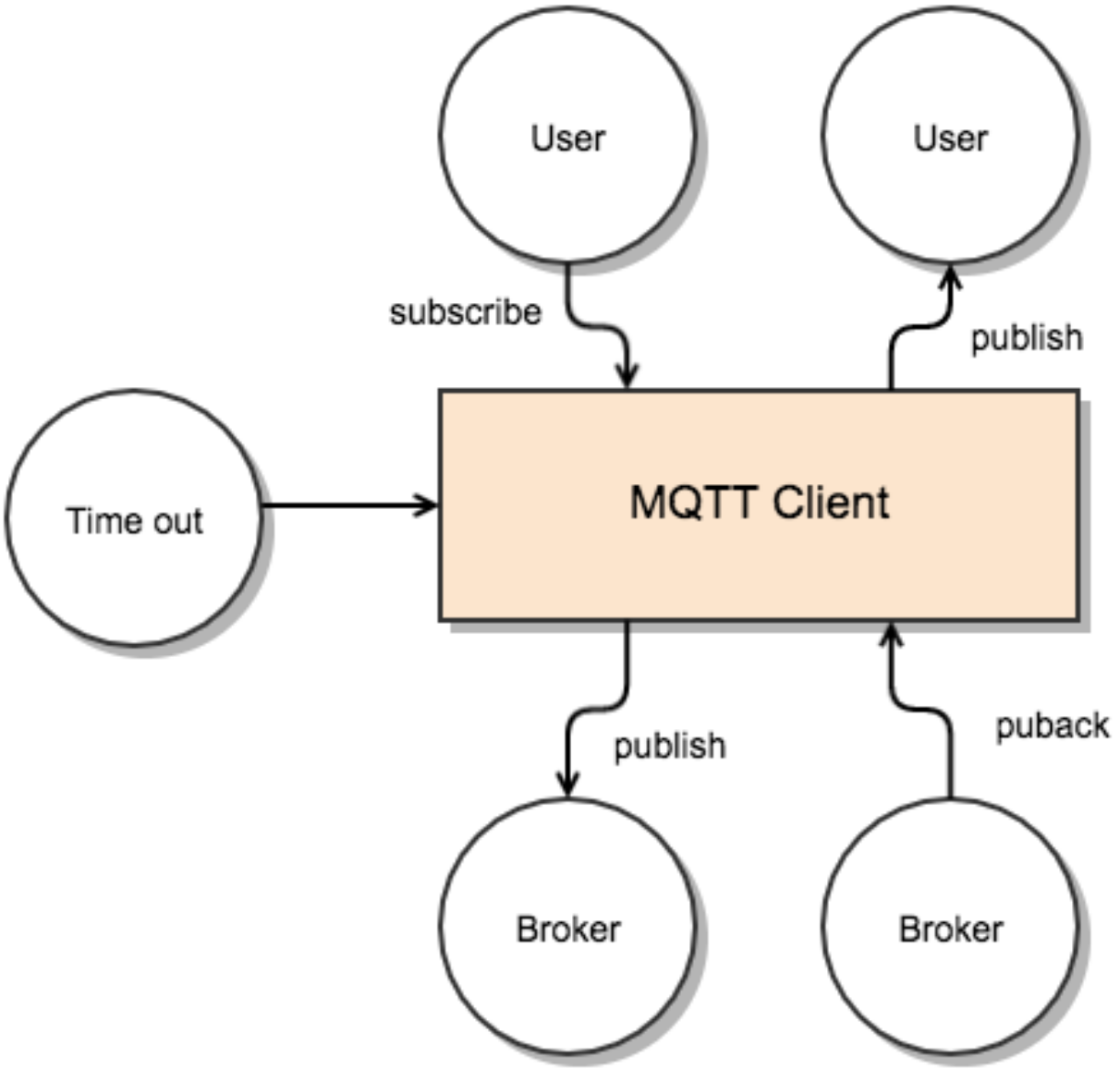


Why another library?

- Blocking
- Dependency injection
- Event notification
- Executor

Injection

Lots going on



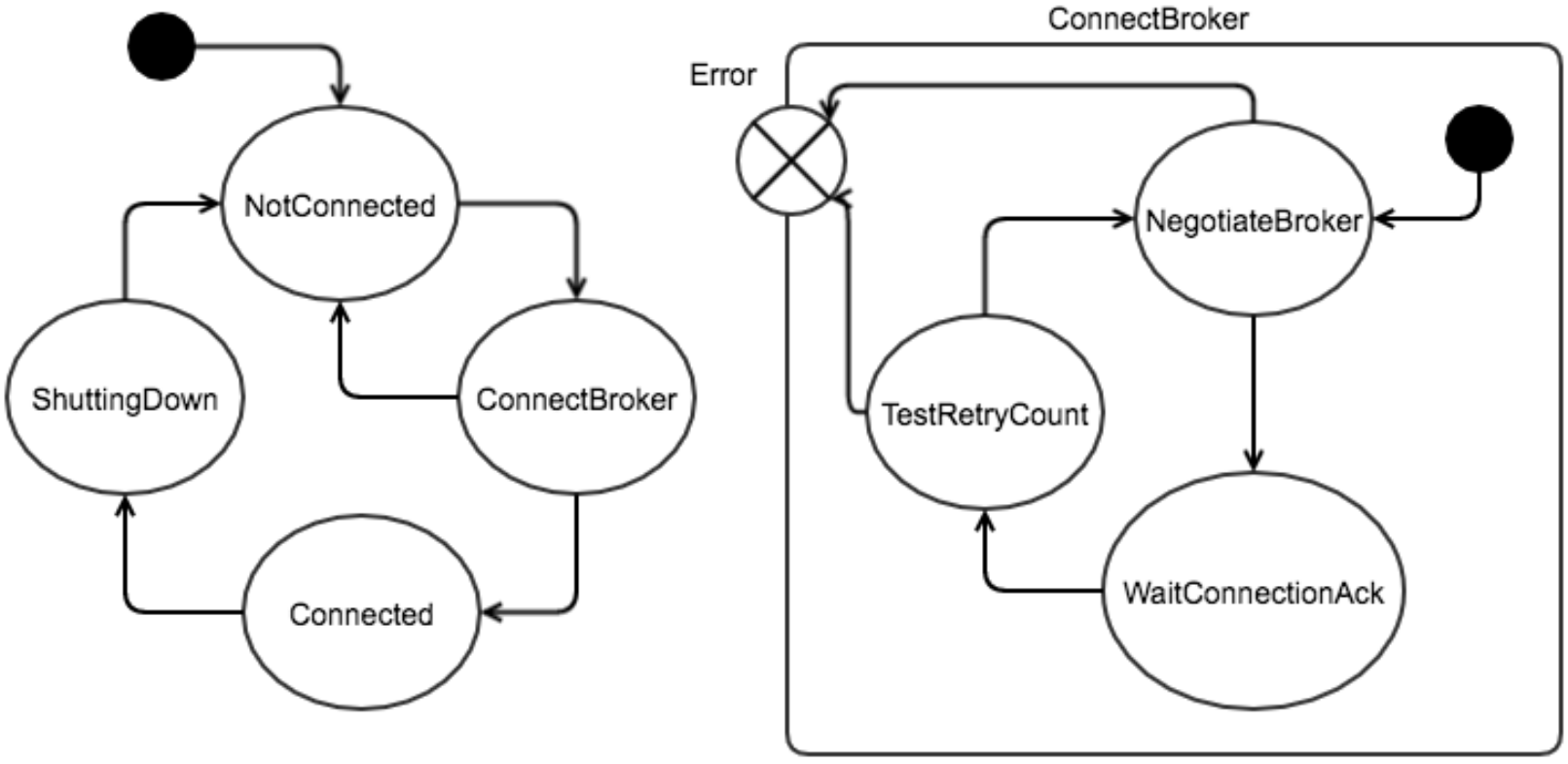
Chain completion handlers

- Establish connection
- Negotiate with broker
- Subscribe

Chain completion handlers?

No!

Hierarchical Finite State-machine

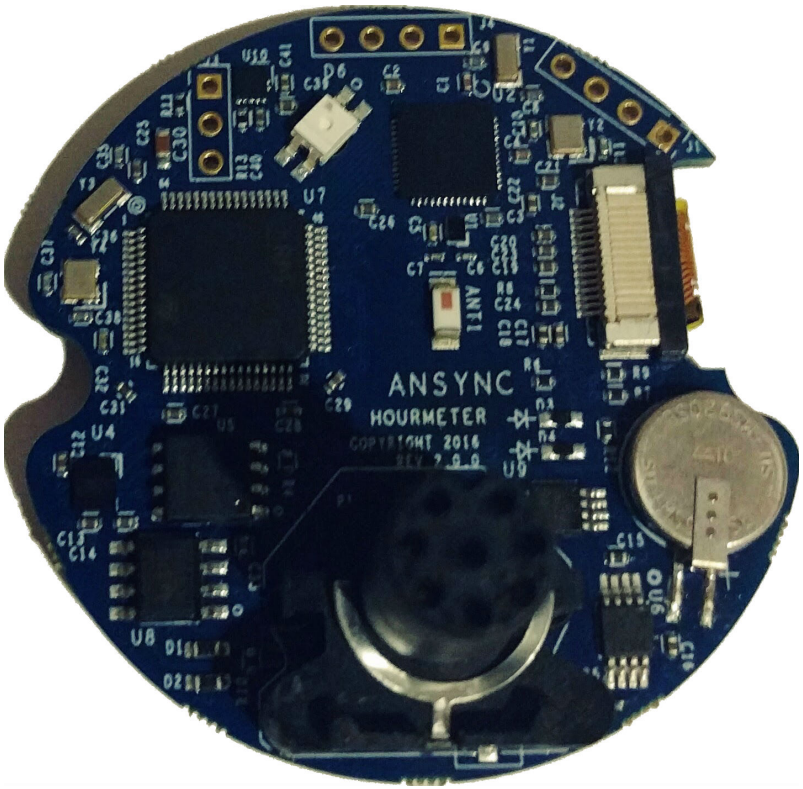


Target Platform

Thank you Ansync Labs!

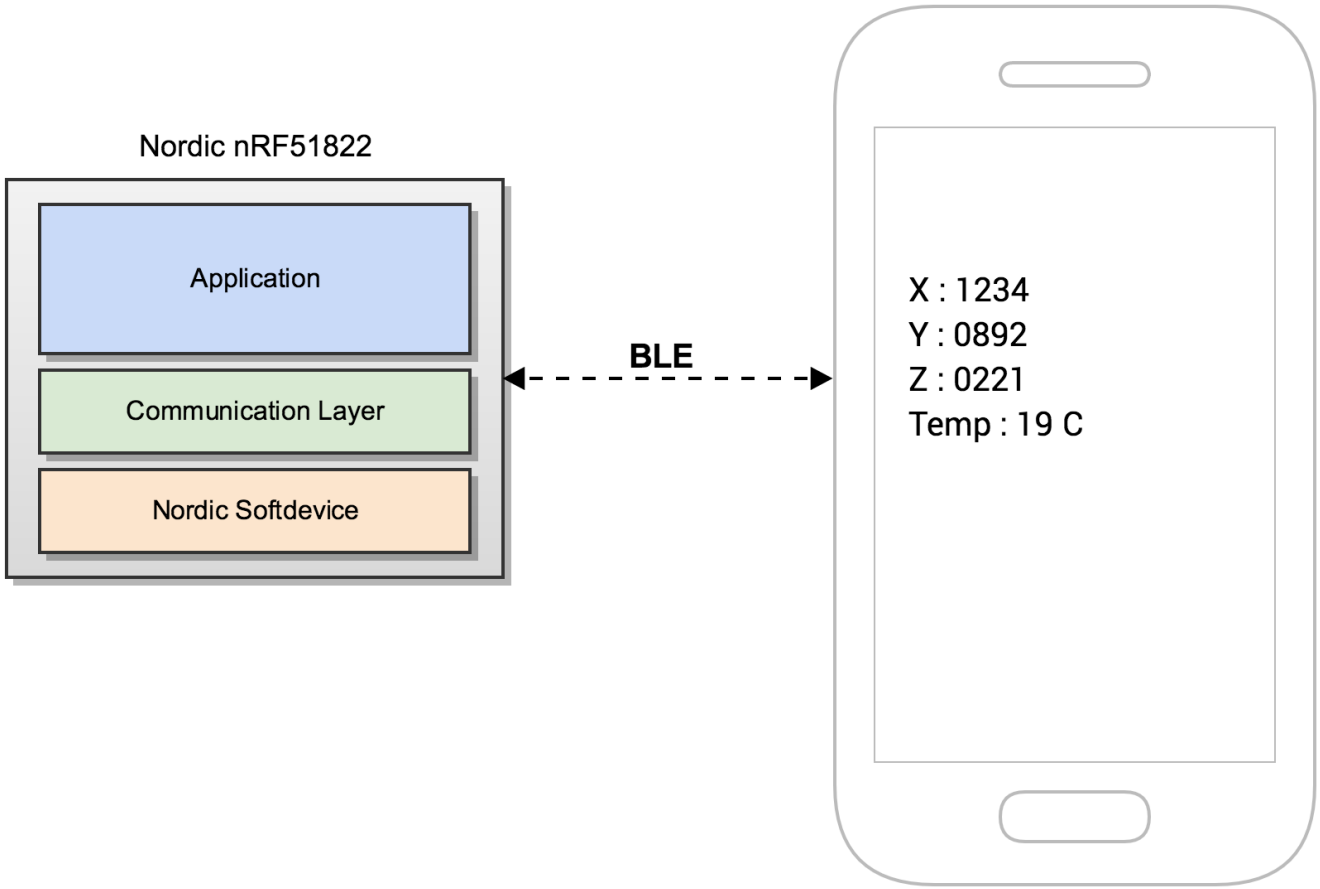
- Nordic nRF51822
- 2.4GHz multiprotocol radio
- 32-bit ARM(R) Cortex(TM) M0
- 128KB embedded flash
- 32KB RAM

http://ansync.com

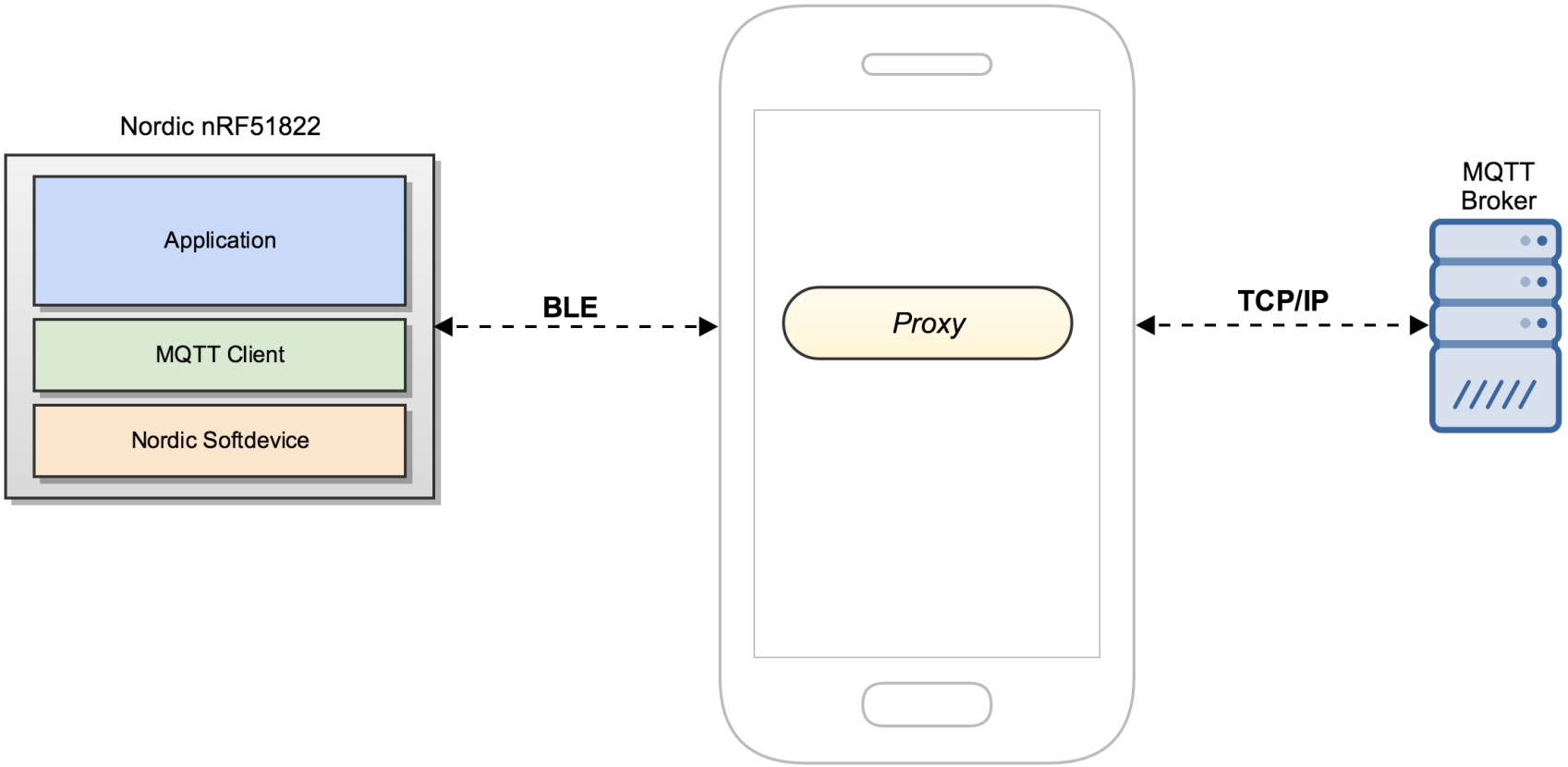


ciere.com

The Original Test App

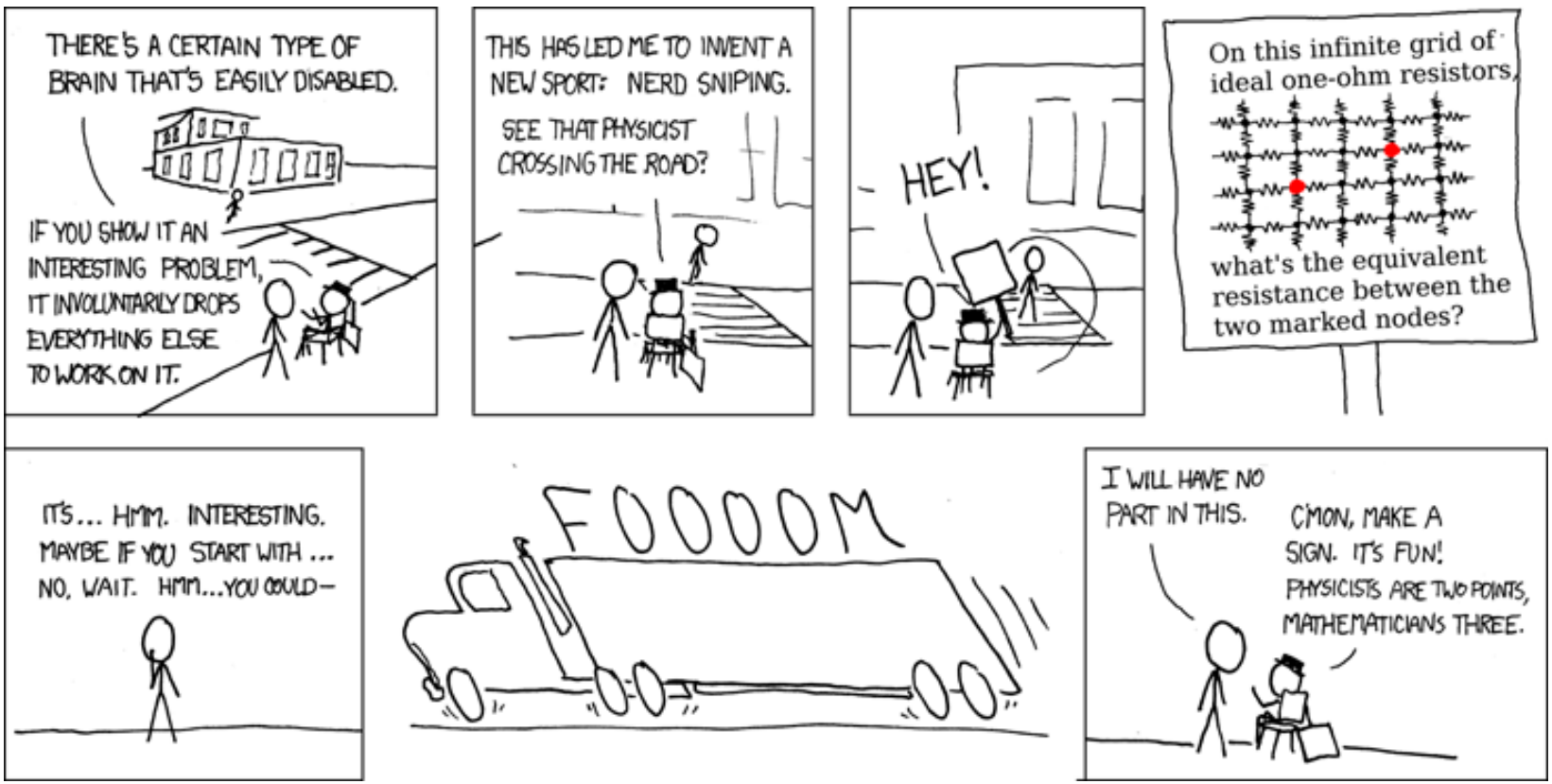


The Goal


Nordic nRF51822
Application
MQTT Client
Nordic Softdevice
BLE
Proxy
TCP/IP
MQTT
Broker

The Sniped

- Agustin K-ballo Berge - wrote the Android App!
- Jeroen Habraken (VeXocide) - X3 contributions



Part II

Diving In

Step 1

Get code to compile with C++.

Cleaning up

```
static int oled_write(uint8_t *bytes, int count) {
  int tx = i2c_master_tx(OLED_I2C_ADDRESS, bytes, count);
  if (tx < 1) return -1;
  return tx;
}
```

```
oled_write((uint8_t []){ 0x00, 0xAF }, 2);
oled_write((uint8_t []){ 0x00, 0x8D, 0x14 }, 3);
oled_write((uint8_t []){ 0x00, 0xA8, 0x0F }, 3);
oled_write((uint8_t []){ 0x00, 0xDA, 0x02 }, 3);
oled_write((uint8_t []){ 0x00, 0x81, 0xFF }, 3);
oled_write((uint8_t []){ 0x00, 0x82, 0xFF }, 3);
oled_write((uint8_t []){ 0x00, 0x83, 0xFF }, 3);
```

Cleaning up

```
int oled_write(uint8_t const * bytes, int count)
{
  int tx = i2c_master_tx( OLED_I2C_ADDRESS
                        , const_cast<uint8_t*>(bytes)
                        , count);
  if (tx < 1) return -1;
  return tx;
}

int oled_write(std::initializer_list<uint8_t> bytes)
{
  return oled_write(bytes.begin(), bytes.size());
}
```

```
oled_write({ 0x00, 0xAF });
oled_write({ 0x00, 0x8D, 0x14 });
oled_write({ 0x00, 0xA8, 0x0F });
oled_write({ 0x00, 0xDA, 0x02 });
oled_write({ 0x00, 0x81, 0xFF });
oled_write({ 0x00, 0x82, 0xFF });
oled_write({ 0x00, 0x83, 0xFF });
```

Baseline

Using: arm-none-eabi-g++ (GCC) 5.1.0

Condition	text	data	bss	total
C99	23576	136	2460	26172
C++14	25404	136	2440	27980

ciere.com

Exceptions

Condition	text	data	bss	total
C99	23576	136	2460	26172
C++14	25404	136	2440	27980
C++14 - fno-rtti	25404	136	2440	27980
C++14 - fno-exceptions	20420	136	2440	22996

Optimize flag `Os`

Condition	text	data	bss	total
C99 O3	23576	136	2460	26172
C++14 O3	25404	136	2440	27980
C++14 O3 - fno-exceptions	20420	136	2440	22996
C99 Os	18984	136	2456	21576
C++14 Os	21436	136	2440	24012
C++14 Os - fno-exceptions	16544	136	2440	19120

Empty main

Condition	text	data	bss	total
C99 Os	720	96	0	816
C++14 Os - fno-exceptions	720	96	0	816

Types

Client to Server

```
struct connect
{
   mqtt::string client_id;
};

struct subscribe
{
   uint16_t packet_id;
   using topic_filter_t = std::tuple<mqtt::string,qos_t>;
   using filters_t = mqtt::vector<topic_filter_t>;
   filters_t filters;
};

struct unsubscribe
{
   uint16_t packet_id;
   mqtt::vector<mqtt::string> filters;
};

struct pingreq
{};

struct disconnect
{
   bool force;
};
```

Types

```
namespace mqtt
{
  using string = std::string;

  template <typename ... T>
  using vector = std::vector<T...>;
}
```

Types

```
namespace mqtt
{
  template < typename Char
           , typename Traits = std::char_traits<Char>>
  using basic_string = std::basic_string< Char, Traits
                                        , my_allocator<Char>>;

  using string = basic_string<char>;

  template <typename T>
  using vector = std::vector<T, my_allocator<T>>;
}
```

Adding string

Change from adding/instantiating `std::string`

Condition	text	data	bss	total
delta	364	0	0	364

mqtt::client

```
template < typename Connection
         , typename Executor
         , typename PublishHandler = mqtt_publish_handler_t >
class client
{
public:
   client( mqtt_string_t identifier
         , Connection & connection, Executor & task_executor);

private:

   detail::client_interface_wrapper< Connection
                                   , Executor
                                   > client_interface_;

   PublishHandler publish_handler_;
   mqtt_string_t mqtt_identifier_;

   detail::client_machine client_machine_;
};
```

mqtt::client

```
template < typename Connection
         , typename Executor
         , typename PublishHandler = mqtt_publish_handler_t >
class client
{
public:
   client( mqtt_string_t identifier
         , Connection & connection, Executor & task_executor);

   void connect(configuration_t);
   void disconnect(bool force=false);

   void subscribe(topic_filter_t filter, handler);
   void subscribe(topic_filter_list_t filters, handler);

   void unsubscribe(topic_filter_t filter, handler);
   void unsubscribe(topic_filter_list_t filters, handler);

   template <typename F>
   void set_publish_handler(F && handler);

   void publish( topic_t topic, payload_t payload
               , qos_t qos, bool retain, handler);

   template <typename T>
   void publish(T, handler);
```

mqtt::client

```
template < typename Connection
         , typename Executor
         , typename PublishHandler = mqtt_publish_handler_t >
class client
{
public:
   client( mqtt_string_t identifier
         , Connection & connection, Executor & task_executor);

   void connect(configuration_t);
   void disconnect(bool force=false);

   void subscribe(topic_filter_t filter, handler);
   void subscribe(topic_filter_list_t filters, handler);

   void unsubscribe(topic_filter_t filter, handler);
   void unsubscribe(topic_filter_list_t filters, handler);

   template <typename F>
   void set_publish_handler(F && handler);

   void publish( topic_t topic, payload_t payload
               , qos_t qos, bool retain, handler);

   template <typename T>
   void publish(T, handler);
```

Construct a `mqtt::client`

```
template <typename Connection, typename Executor, typename PublishHandler>
client<Connection,Executor,PublishHandler>::
client( mqtt_string_t identifier
      , Connection & connection, Executor & task_executor)
   : client_interface_(connection, task_executor)
   , mqtt_identifier_(identifier)
{
   initialize_submachines(client_machine_, &client_interface_);
   client_machine_.start();
}
```

Using boost::msm

```
struct client_machine_ : public machine_base<client_machine_>
{

};

using client_machine = boost::msm::back::state_machine<client_machine_>;
```

Using boost::msm

```
struct transition_table : TransitionTable<
//     Start                Event                          Next                Action              Guard
//   +--------------------+------------------------------+-------------------+-------------------+----------
Row <  NotConnected       , event::connect               , ConnectBroker     , none              , none   >,
Row <  ConnectBroker      , none                         , Connected         , none              , none   >,
Row <  Connected          , event::publish_out           , none              , send_packet       , none   >,
Row <  Connected          , event::subscribe             , none              , send_packet       , none   >,
Row <  Connected          , event::unsubscribe           , none              , send_packet       , none   >,
Row <  Connected          , event::connect               , none              , none              , none   >,
Row <  Connected          , event::disconnect            , ShuttingDown      , none              , none   >,
Row <  ShuttingDown       , event::shutdown_timeout      , NotConnected      , none              , none   >
//   +--------------------+------------------------------+-------------------+-------------------+----------
>{};
```

Using boost::msm

```
struct NotConnected : public State
{
    template <class Event, class FSM>
    void on_entry(Event const &, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTED);
    }
};

struct Connected : public State
{
    template< class Event, class FSM >
    void on_entry(Event const&, FSM& fsm)
    {
        fsm.client_->update_connection_status(CONNECTED);
    }

    template< class Event, class FSM >
    void on_exit(Event const&, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTING);
    }
};

struct ShuttingDown : public State
{
    // defer connect events until the next state
    using deferred_events = mpl::vector<event::connect>;
};
```

Using boost::msm

```
struct NotConnected : public State
{
    template <class Event, class FSM>
    void on_entry(Event const &, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTED);
    }
};

struct Connected : public State
{
    template< class Event, class FSM >
    void on_entry(Event const&, FSM& fsm)
    {
        fsm.client_->update_connection_status(CONNECTED);
    }

    template< class Event, class FSM >
    void on_exit(Event const&, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTING);
    }
};

struct ShuttingDown : public State
{
    // defer connect events until the next state
    using deferred_events = mpl::vector<event::connect>;
};
```

Using boost::msm

```
struct NotConnected : public State
{
    template <class Event, class FSM>
    void on_entry(Event const &, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTED);
    }
};

struct Connected : public State
{
    template< class Event, class FSM >
    void on_entry(Event const&, FSM& fsm)
    {
        fsm.client_->update_connection_status(CONNECTED);
    }

    template< class Event, class FSM >
    void on_exit(Event const&, FSM & fsm)
    {
        fsm.client_->update_connection_status(DISCONNECTING);
    }
};

struct ShuttingDown : public State
{
    // defer connect events until the next state
    using deferred_events = mpl::vector<event::connect>;
};
```

Using boost::msm

```
struct transition_table : TransitionTable<
//     Start                Event                          Next                Action              Guard
//   +--------------------+------------------------------+-------------------+-------------------+----------
Row <  NotConnected       , event::connect               , ConnectBroker     , none              , none   >,
Row <  ConnectBroker      , none                         , Connected         , none              , none   >,
Row <  Connected          , event::publish_out           , none              , send_packet       , none   >,
Row <  Connected          , event::subscribe             , none              , send_packet       , none   >,
Row <  Connected          , event::unsubscribe           , none              , send_packet       , none   >,
Row <  Connected          , event::connect               , none              , none              , none   >,
Row <  Connected          , event::disconnect            , ShuttingDown      , none              , none   >,
Row <  ShuttingDown       , event::shutdown_timeout      , NotConnected      , none              , none   >
//   +--------------------+------------------------------+-------------------+-------------------+----------
>{};
```

Using boost::msm

```
struct send_packet
{
   template<class Fsm, class SourceState, class TargetState>
   void operator()( event::publish_out const& evt, Fsm& fsm
                  , SourceState&, TargetState& )
   {
      fsm.client_->send(evt.publish);
   }

   template<class Fsm, class SourceState, class TargetState>
   void operator()( event::subscribe const& evt, Fsm& fsm
                  , SourceState&, TargetState& )
   {
      fsm.client_->send(evt.subscribe);
   }

   template<class Fsm, class SourceState, class TargetState>
   void operator()( event::unsubscribe const& evt, Fsm& fsm
                  , SourceState&, TargetState& )
   {
      fsm.client_->send(evt.unsubscribe);
   }
};
```

Using boost::msm

```
struct client_machine_ : public machine_base<client_machine_>
{

   using submachines = Submachines<ConnectBroker>;
   using initial_state = NotConnected;

};

using client_machine = boost::msm::back::state_machine<client_machine_>;
```

FSM Helpers

```
template <typename ...T>
using TransitionTable = boost::mpl::vector<T...>;

template <typename ...T>
using Submachines = meta::meta_list<T...>;
```

```
template <typename Derived>
struct machine_base : public msm::front::state_machine_def<Derived>
{
   mqtt::detail::client_interface * client_ = nullptr;
};
```

FSM Helpers

```
template <typename ...T>
using TransitionTable = boost::mpl::vector<T...>;

template <typename ...T>
using Submachines = meta::meta_list<T...>;
```

```
template <typename Derived>
struct machine_base : public msm::front::state_machine_def<Derived>
{
   mqtt::detail::client_interface * client_ = nullptr;
};
```

Type Erased Interface

```
struct client_interface
{
   virtual void send_to_broker(uint8_t const * data, uint16_t length) = 0;
   virtual void receive_from_broker(/*...*/) = 0;
   virtual void queue_task(/*...*/) = 0;
   virtual void update_connection_status(/*...*/) = 0;

   template <typename Packet>
   void send(Packet const & packet)
   {
      auto buffer = serialize(packet);
      send_to_broker(pointer(buffer), size(buffer));
   }
};
```

Type Erased Interface

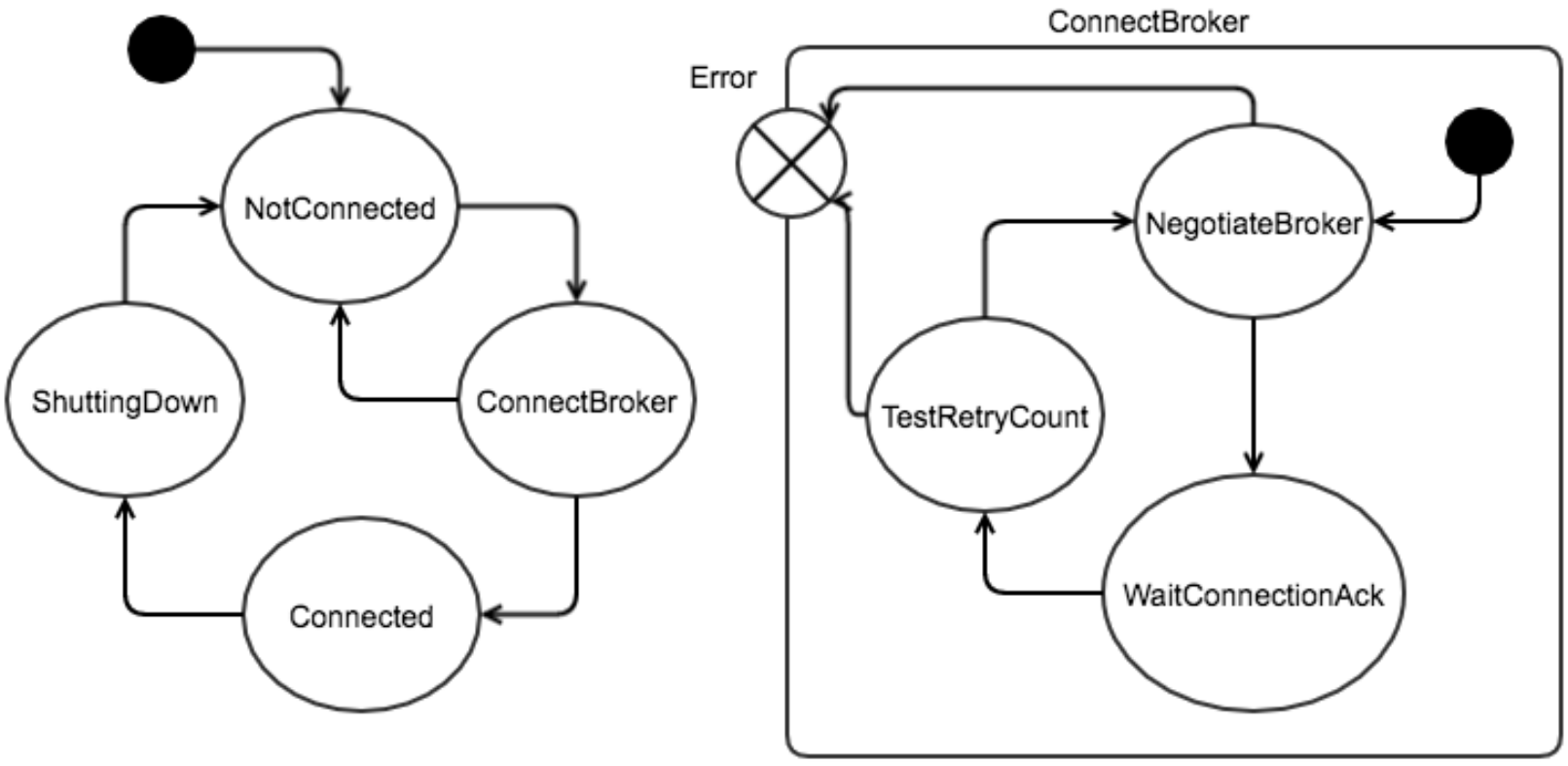
```
template <typename Connection, typename Executor>
struct client_interface_wrapper : client_interface
{
   client_interface_wrapper(Connection & broker, Executor & executor)
      : broker_(broker)
      , executor_(executor)
   {}

   virtual void send_to_broker( uint8_t const * data
                              , uint16_t length) override
   {
      broker_.send(data, length);
   }

private:
   Connection & broker_;
   Executor & executor_;
};
```

Hierarchical Finite State-machine

Hierarchical Finite State-machine



Constructing `mqtt::client`

```
template <typename Connection, typename Executor, typename PublishHandler>
client<Connection,Executor,PublishHandler>::
client( mqtt_string_t identifier
      , Connection & connection, Executor & task_executor)
   : client_interface_(connection, task_executor)
   , mqtt_identifier_(identifier)
{
   initialize_submachines(client_machine_, &client_interface_);
   client_machine_.start();
}
```

Initializing the Hierarchy

Technique lifted from **`cierelabs::ladon`**

```
template <typename Machine, typename Interface>
void initialize_submachines( Machine & machine
                           , Interface * interface )
{
   machine.client_ = interface;
   initialize_submachine_impl<Machine,Interface> impl(machine,interface);
   decend<Machine>::apply(&impl);
}
```

Initializing the Hierarchy

```
template <typename MSMBackEnd, typename Interface>
struct initialize_submachine_impl
{
   initialize_submachine_impl(MSMBackEnd & machine, Interface * interface)
      : machine_(machine)
      , interface_(interface)
   {}

   template <typename SubMach>
   void operator()(SubMach const &)
   {
      SubMach* submachine = machine_. template get_state<SubMach*>();
      submachine->client_ = interface_;

      initialize_submachine_impl<SubMach,Interface> impl( *submachine
                                                        , interface_ );
      decend<SubMachine>::apply(&impl);
   }

   MSMBackEnd & machine_;
   Interface * interface_;
};
```

Initializing the Hierarchy

```
template <typename Machine, typename Enable = void>
struct decend
{
   template<typename T>
   static void apply(T*) {}
};
```

```
template <typename Machine>
struct decend< Machine
             , std::enable_if_t<meta::size_v<typename Machine::submachines>
                                != 0> >
{
   template<typename T>
   static void apply(T* t)
   {
      meta::for_each<typename Machine::submachines>(*t);
   }
};
```

Initializing the Hierarchy

```
template <typename Machine, typename Enable = void>
struct decend
{
   template<typename T>
   static void apply(T*) {}
};
```

```
template <typename Machine>
struct decend< Machine
             , std::enable_if_t<meta::size_v<typename Machine::submachines>
                                != 0> >
{
   template<typename T>
   static void apply(T* t)
   {
      meta::for_each<typename Machine::submachines>(*t);
   }
};
```

Wut! Another TMP Library?

```
template <typename ... T>
struct meta_list {};
```

size

```
template <typename List>
struct size
{};

template <typename ... T>
struct size<meta_list<T...>>
{
   static const std::size_t value = sizeof...(T);
};
```

```
template< class T >
constexpr std::size_t size_v = size<T>::value;
```

size

```
template <typename List>
struct size
{};

template <typename ... T>
struct size<meta_list<T...>>
{
   static const std::size_t value = sizeof...(T);
};
```

```
template< class T >
constexpr std::size_t size_v = size<T>::value;
```

for_each

```
namespace detail
{
   struct do_nothing
   {
      template <typename ... Args>
      do_nothing(Args && ...){}
   };

   template <typename ...T, typename Func>
   Func for_each(meta_list<T...> &&, Func && f)
   {
      do_nothing((f(T{}),0)...);
      return f;
   }
}
```

```
template <typename List, typename Func>
Func for_each(Func && f)
{
   return detail::for_each(List{}, std::forward<Func>(f));
}
```

for_each

```
namespace detail
{
   struct do_nothing
   {
      template <typename ... Args>
      do_nothing(Args && ...){}
   };

   template <typename ...T, typename Func>
   Func for_each(meta_list<T...> &&, Func && f)
   {
      do_nothing((f(T{}),0)...);
      return f;
   }
}
```

```
template <typename List, typename Func>
Func for_each(Func && f)
{
   return detail::for_each(List{}, std::forward<Func>(f));
}
```

Cost of `initialize_submachines`

Condition	text	data	bss	total
Just the parent	0	0	0	0
With submachine	28	0	0	28

Cost of `initialize_submachines`

Why the complexity??

Easy for user!

```
struct client_machine_ : public machine_base<client_machine_>
{

   using submachines = Submachines<ConnectBroker>;
   using initial_state = NotConnected;

};

using client_machine = boost::msm::back::state_machine<client_machine_>;
```

Types

```
enum class qos_t : uint8_t
{
    QOS0    = 0x00,
    QOS1    = 0x01,
    QOS2    = 0x02,
    FAILURE = 0x80,
};
```

Bidirectional

```
struct publish
{
    bool dup = false;
    uint8_t qos = 0x00;
    bool retain = false;
    mqtt::string topic_name;
    uint16_t packet_id;
    mqtt::vector<uint8_t> payload;
};

struct puback
{
    uint16_t packet_id;
};
```

Types

```
enum class qos_t : uint8_t
{
   QOS0    = 0x00,
   QOS1    = 0x01,
   QOS2    = 0x02,
   FAILURE = 0x80,
};
```

Bidirectional

```
struct publish
{
   bool dup = false;
   uint8_t qos = 0x00;
   bool retain = false;
   mqtt::string topic_name;
   uint16_t packet_id;
   mqtt::vector<uint8_t> payload;
};

struct puback
{
   uint16_t packet_id;
};
```

Types

Server to Client

```
struct connack
{
   bool session_pressent;
   uint8_t response_code;
};

struct suback
{
   uint16_t packet_id;
   using qos_obtained_t = mqtt::vector<qos_t>;
   qos_obtained_t qos_obtained;
};

struct unsuback
{
   uint16_t packet_id;
};

struct pingresp
{};
```

Types

Incomming control messages:

```
struct control_packet
   : public x3::variant< puback
                       , connack
                       , suback
                       , unsuback
                       , pingresp >
{
   using base_type::base_type;
   using base_type::operator=;
};
```

Adapted Types

```
BOOST_FUSION_ADAPT_STRUCT(
    cierelabs::mqtt::packet::puback,
    (uint16_t, packet_id)
)

BOOST_FUSION_ADAPT_STRUCT(
    cierelabs::mqtt::packet::connack,
    (bool    , session_pressent)
    (uint8_t, response_code)
)

BOOST_FUSION_ADAPT_STRUCT(
    cierelabs::mqtt::packet::suback,
    (uint16_t, packet_id)
    (cierelabs::mqtt::packet::suback::qos_obtained_t, qos_obtained)
)

BOOST_FUSION_ADAPT_STRUCT(
    cierelabs::mqtt::packet::unsuback,
    (uint16_t, packet_id)
)

BOOST_FUSION_ADAPT_STRUCT(
    cierelabs::mqtt::packet::publish,
    (mqtt::string    , topic_name)
    (uint16_t, packet_id)
    (mqtt::vector<uint8_t>, payload)
)
```

Parse it!

```
bool keepPacketizing = true;
while(keepPacketizing)
{
    switch(parsePhase_)
    {
        case ParsePhase::HEADER_BYTE:
            keepPacketizing = readHeader();
            break;
        case ParsePhase::LENGTH:
            keepPacketizing = readLength();
            break;
        case ParsePhase::BODY:
            keepPacketizing = readBody();
            break;
    }
}
```

Parse it!

```
bool MQTTPacketizer::readHeader()
{
    if(!data_.empty())
    {
        currentHeader_ = data_.front();
        data_.pop_front();
        parsePhase_ = ParsePhase::LENGTH;
        return true;
    }

    return false;
}
```

Parse it!

```
void MQTTPacketizer::formPacket()
{
    switch(currentHeader_ & 0xf0)
    {
        case MQTTPacketType::PUBLISH:
            receivePacket(makePublishPacket());
            break;

        case MQTTPacketType::CONNACK:
            receivePacket(makeConnackPacket());
            break;

        case MQTTPacketType::PUBACK:
            receivePacket(makePubackPacket());
            break;

        case MQTTPacketType::SUBACK:
            receivePacket(makeSubackPacket());
            break;
```

Parse it!

```
packet::suback MQTTPacketizer::makeSubackPacket()
{
   packet::suback packet;
   packet.packetID = getUint16();

   int responseCount = currentLength_ - 2;
   for(int i=0; i<responseCount; ++i)
   {
      packet.qosObtained.push_back(
         static_cast<packet::suback::Code>(getUint8())
         );
   }

   return packet;
}
```

boost::spirit ??

```
auto const control_packet_def =
      omit[byte_(0x40)] >> puback
    | omit[byte_(0x10)] >> publish
    | omit[byte_(0x20)] >> connack
    | omit[byte_(0x90)] >> suback
    | omit[byte_(0xb0)] >> unsuback
    | omit[byte_(0xc0)] >> attr(packet::pingresp{})
    ;
```

boost::spirit ??

```
auto const qos_def =
      omit[byte_(0x00)] >> attr(packet::qos_t::QOS0)
    | omit[byte_(0x01)] >> attr(packet::qos_t::QOS1)
    | omit[byte_(0x02)] >> attr(packet::qos_t::QOS2)
    | omit[byte_(0x80)] >> attr(packet::qos_t::FAILURE)
    ;
```

```
auto const suback_def =
       skip_encoded_length
    >> big_word
    >> *qos
    ;
```

boost::spirit ??

```
auto const qos_def =
      omit[byte_(0x00)] >> attr(packet::qos_t::QOS0)
    | omit[byte_(0x01)] >> attr(packet::qos_t::QOS1)
    | omit[byte_(0x02)] >> attr(packet::qos_t::QOS2)
    | omit[byte_(0x80)] >> attr(packet::qos_t::FAILURE)
    ;
```

```
auto const suback_def =
      skip_encoded_length
    >> big_word
    >> *qos
    ;
```

boost::spirit ??

```
auto const puback_def =
      skip_encoded_length
   >> big_word
   ;

auto const connack_def =
      skip_encoded_length
   >> (   omit[byte_(0x01)] >> attr(true)
        | attr(false) )
   >> byte_
   ;

auto const unsuback_def =
      skip_encoded_length
   >> big_word
   ;

auto const publish_def =
      skip_encoded_length
   >> big_word
   ;
```

boost::spirit

Grammar (composed parsers) and usage:

Just over 3KB !

Templates are not evil

Don’t try and be smarter than your compiler

struct to bytes

```
template <typename T>
auto serialize(T && packet)
{
    auto variable_length = get_variable_length(packet);
    // packet size : variable length + control header + encoded length size
    uint16_t packet_size = variable_length + 1 +
                           (int)(variable_length/128) + 1;

    auto packet_data = get_packet_buffer(packet_size);
    serialize_impl( packet
                  , std::back_inserter(packet_data)
                  , variable_length);

    return packet_data;
}
```

struct to bytes

```
template <typename Iterator>
Iterator serialize_impl( packet::subscribe const & subscribe
                       , Iterator iter
                       , int variable_length)
{
   // control header
   *iter++ = 0x82;

   encode_length(variable_length, iter);
   encode(subscribe.packet_id, iter);

   for(auto & topic : subscribe.filters)
   {
      encode(std::get<0>(topic), iter);
      *iter++ = static_cast<uint8_t>(std::get<1>(topic));
   }
   return iter;
}
```

struct to bytes

```cpp
template <typename T>
auto serialize(T && packet)
{
    auto variable_length = get_variable_length(packet);
    // packet size : variable length + control header + encoded length size
    uint16_t packet_size = variable_length + 1 +
                           (int)(variable_length/128) + 1;

    auto packet_data = get_packet_buffer(packet_size);
    serialize_impl( packet
                  , std::back_inserter(packet_data)
                  , variable_length);

    return packet_data;
}

constexpr uint16_t get_variable_length(packet::pingreq const &)
{
    return 0;
}

template <typename Iterator>
Iterator serialize_impl(packet::pingreq const &, Iterator iter, int)
{
    copy({0xc0, 0x00}, iter);
    return iter;
}
```

struct to bytes

```
template <typename T>
auto serialize(T && packet)
{
   auto variable_length = get_variable_length(packet);
   // packet size : variable length + control header + encoded length size
   uint16_t packet_size = variable_length + 1 +
                          (int)(variable_length/128) + 1;

   auto packet_data = get_packet_buffer(packet_size);
   serialize_impl( packet
                 , std::back_inserter(packet_data)
                 , variable_length);

   return packet_data;
}

constexpr uint16_t get_variable_length(packet::pingreq const &)
{
   return 0;
}

template <typename Iterator>
Iterator serialize_impl(packet::pingreq const &, Iterator iter, int)
{
   copy({0xc0, 0x00}, iter);
   return iter;
}
```

struct to bytes

```cpp
template <typename T>
auto serialize(T && packet)
{
   auto variable_length = get_variable_length(packet);
   // packet size : variable length + control header + encoded length size
   uint16_t packet_size = variable_length + 1 +
                          (int)(variable_length/128) + 1;

   auto packet_data = get_packet_buffer(packet_size);
   serialize_impl( packet
                 , std::back_inserter(packet_data)
                 , variable_length);

   return packet_data;
}

constexpr uint16_t get_variable_length(packet::pingreq const &)
{
   return 0;
}

template <typename Iterator>
Iterator serialize_impl(packet::pingreq const &, Iterator iter, int)
{
   copy({0xc0, 0x00}, iter);
   return iter;
}
```

Part III

Conclusion

A few thoughts

- Templates are not evil
- Know what you are doing with templates
- Determine your level of *“embedded”*
- Use `std` containers to start
- Don’t out-think the compiler!

ciere.com

A few thoughts

- Templates are not evil
- Know what you are doing with templates
- Determine your level of *“embedded”*
- Use `std` containers to start
- Don’t out-think the compiler!

A few thoughts

- Templates are not evil
- Know what you are doing with templates
- Determine your level of *“embedded”*
- Use `std` containers to start
- Don’t out-think the compiler!

A few thoughts

- Templates are not evil
- Know what you are doing with templates
- Determine your level of *“embedded”*
- Use **`std`** containers to start
- Don’t out-think the compiler!

ciere.com

A few thoughts

- Templates are not evil
- Know what you are doing with templates
- Determine your level of *“embedded”*
- Use **`std`** containers to start
- Don’t out-think the compiler!

What is left?

- Decide on memory/container strategy
- Extension points
- Documentation
- Case studies
- Testing

A Name!

Name?

Macchiato

ma**q**i**att**o

Where do I find it?

https://github.com/cierelabs/maqiatto

Part IV

Bonus