

Units background

Summary of past decisions

- Agreement that ns-3 would benefit from strongly typed units and quantities
- Agreement to take an interim step in the wifi module to introduce "weak types" (aliasing of C++ double type)
- Maintenance and usability tradeoffs exist among the possible solutions (existing libraries vs. custom)
- Two alternatives have been developed with ns-3 code: Jiwoong Lee's custom library (MR [!1887](#)) and adaptation of existing nholthaus/units library (MR [!2085](#))
- In August 2024, we agreed to further explore nholthaus-units on an experimental branch, with these open questions:
 - Compatibility with Python API
 - Extensions to unsupported units (dBr_u, dBm_per_MHz)
 - Port more of the existing code

Updates in early 2025

- Some additional development of Jiwoong's approach has occurred (not reflected in the current GitLab.com MR)
- Some discussion in the tracker about whether we would alias nholthous/units types into ns3 namespace; i.e.

```
namespace ns3{  
using Hz_t = units::frequency::hertz_t;  
} // namespace ns3
```

 - Current branch supports this, aligning with strong type (e.g., dBm_u -> dBm_t)
- Gabriel experimented with the [Python API](#) and options to inject into ns3 namespace
- Sebastien suggested to fully port both power and frequency variables in the wifi module to fully evaluate suitability

Recent updates

- Converted all power quantities from weak to strong type in the src/wifi directory only
 - dBr_u, dBm_u, dB_u, dBW_u, Watt_u
 - At interfaces to the wifi module (propagation, spectrum, Wi-Fi trace sources), the double value is still used
 - This was very tedious work (~2600 LOC changed) and Claude Code was surprisingly poor at assisting (I also tried Google Gemini)
 - Reviewed/updated 47 comments in MR !2085
- Most tests pass— all remaining test failures are not units related but are problems with the tests and can be reproduced outside of this branch
 - Try test.py after restricting the build to `–enable-modules=wifi`
- Added support for 'dBm_per_MHz' and 'dBm_per_Hz' units, which were not in the nholthaus/units library

Highlights

- Run and read the units-example.cc program to get a flavor for it
- Key core files to look at:
 - src/core/model/units.h
 - src/core/model/units-nholthaus.h
 - src/core/model/decibel.{cc,h}
 - src/core/mode/dbm.{cc,h}
- Good example wifi model file to look at the diff:
 - [wifi-phy.cc](#)

My impressions

- This still seems to me to be the best way forward
 - This library has been in wide use in the C++ ecosystem
 - I think that it would be low friction for users and maintainers
 - Unit errors are caught at compile time (except for callback signature mismatches) and tend to be pretty obvious
 - Highest payoff is likely to be in avoiding logarithmic/linear mistakes for power quantities
 - Compatible with command-line arguments and attribute values
- Sebastien's interim weak types port (double type to aliases such as dB_u) was very valuable in helping to port
- A major tedious job remains to convert the rest of the codebase

Next steps

- Frequency quantities (primarily MHz) is the other significant port remaining for wifi module
 - Should be easier than power quantities
 - I already tested some frequency examples in the units-example.cc
 - There are some energy related port (e.g., current in amperes) also needed
- Rebase to latest ns-3-dev (and port new wifi code that was added shortly before the ns-3.46 release)
- Enable all modules in the build
 - mesh module needs conversion, and some examples and tests are not covered by --enable-modules=wifi
- Start to convert lte, zigbee, lr-wpan, spectrum, propagation, nr, ... ?

Open issues

- Would we eventually want to break trace source backward compatibility to impose strongly typed units on those parameters?
- Upstream (nholthaus/units) has basically gone unmaintained; is it an issue for us?
 - I think not, but just mentioning this
- Would we port the boost::units (length) code in src/core?
 - I think yes, since it is lightly used
- Jiwoong has an open comment on [ARM macros](#) that I did not address

Backup

(slides from August 2024 meeting)

Units background

- There seems to be consensus for adopting strongly-typed units and quantities
- No consensus reached for whether to adopt double type aliases (weak types) as an interim step
 - Sebastien's [MR proposal !2068](#)
- No consensus on the actual implementation
 - Jiwoong proposed a new [units library](#) and volunteered to maintain it
 - I have been working with [wrapping nholthaus/units](#) library as a possible alternative
 - Peter expressed interest in aligning with [mp-units](#) because of possible future standard library inclusion

My concerns with various proposals

- The choice involves a complicated tradeoff between safety, usability, and API consistency
- Writing and maintaining a custom units library is a lot of work
- mp-units seems to have some current momentum but appears to be many years out, and relies heavily on C++ 20/23 features and concepts
- Of widely adopted libraries, only nholthaus/units has support for logarithmic units (power) needed in ns-3
 - Good comparison summary [here](#)

SI units

- Pros
 - Uses mainly basic C++ constructs (good for new users, Python bindings)
 - Jiwoong volunteered to maintain it
- Cons
 - Not aligned with ns3::Time design (API discontinuity)
 - Does not conform to ns-3 naming style
 - Combinatorial explosion of operators/converters possible

nholthaus/units

- Pros
 - Drops into core module pretty easily
 - Probably much less maintenance burden
 - Seems to work as advertised
 - Unit conversions happen implicitly and automatically at compile time
- Cons
 - Likely to cause problems (be incompatible with?) Python bindings
 - Does not align with ns3::Time design (API discontinuity)
 - Creates a bit more friction for users (multiple namespaces, template compilation errors)

Weak types

- Do not see enough benefit over the cost of supporting
- Main benefits appear to be:
 - 1) "better-than-nothing" until we have strong types
 - 2) unit hints for IDEs for trace source parameters that remain double type (for backward compatibility)
- Costs are additional typedefs to manage and distinguish from strong units
 - user question: when should I use `meter_t` vs. `Meter` vs. `double`?
 - aliases will bleed into ns3 namespace in general and may be long-lasting
- Variable naming may provide similar/better benefits
 - e.g., spotting errors in GitLab.com code reviews

Extending the ns-3 Time pattern

- ns3::Time class adopts the paradigm that the **dimension is the type**, with a (configurable) **base unit**, an **underlying integer type**, **converters to export/import double type**, and **scaling converters to print different units**
- We could do similar with Frequency, Power, LogPower, Energy, etc. as needed; e.g.

```
Frequency f = MegaHertz(10) + MilliHertz(20);  
std::cout << f.GetHertz() << std::endl;  
prints: 10000000.02  
std::cout << f.As(Frequency::KHZ) << std::endl;  
prints: +1000.00002kHz
```

My preferences

- No change for ns-3.43 release (Sept)
- Investigate whether nholthaus/units can work with Python
- If yes? See if people can live with nholthaus/units
 - Develop and maintain branch based on wrapping nholthaus/units, to merge immediately after ns-3.43
 - Copy units.h into src/core/model, as in current MR
 - Port Wi-Fi power and frequency types for the initial MR
 - Convert to double type at wifi module boundaries
 - Use weak types MR as a porting guide (the compiler will tell us if we missed any)
 - Keep ns3::Time and ns3::DataRate for backward compat.
 - Eventually replace Boost::Length (only lightly used)
 - Grow outward (other modules, other types) from there

My preferences (cont.)

- If no? Develop custom solution for power and frequency units that is aligned with ns3::Time
 - Use weak types MR as a porting guide

Peter's proposal

- Adopt weak types now in wifi module only, not in user code or other modules for now
- Further explore nholthaus/units in a branch
 - For ns-3.43 (October?) if it works out

nholthaus/units

- **Types are the units themselves** (e.g., Meters)
- Types of different units (scales), but the same **dimension** (e.g., Time, Power), are related
- Types wrap underlying **double values**
- **Unit conversions happen implicitly and automatically at compile time**
 - Disallowed conversions will not compile

```
../src/wifi/model/wifi-phy.cc:200:93: required from here
/home/tomh/temp4/ns-3-units/src/core/model/attribute-accessor-helper.h:477:39: error: cannot convert
'double' to 'const dBm_t&' {aka 'const units::unit_t<units::unit<std::ratio<1, 1000>, units::unit<
std::ratio<1>, units::base_unit<std::ratio<2>, std::ratio<1>, std::ratio<-3> > >, std::ratio<0, 1>, s
td::ratio<0, 1> >, double, units::decibel_scale&'}
477 |         v->Set((object->*m_getter)());
    |               ~~~~~^~~~~~
    |               |
    |             double
In file included from ../build/include/ns3/attribute-helper.h:1,
                 from ../src/wifi/model/wifi-mode.h:26,
                 from ../src/wifi/model/wifi-tx-vector.h:24,
                 from ../src/wifi/model/wifi-ppdu.h:23,
                 from ../src/wifi/model/phy-entity.h:27,
                 from ../src/wifi/model/wifi-phy.h:24,
                 from ../src/wifi/model/wifi-phy.cc:21:
```

Steps to support nholthaus/units

- Added units.h to src/core/model, with linting guards
 - had to also convert from DOS to Unix file type
 - at the moment, I am extending units.h with ns-3 specific things, but could move them to another header (e.g. units-ns3.h)
- Added two new macros for Attribute helpers, to enable support for DoubleValue configuration and for bounds checking
- CommandLine needed stream extraction operators (operator>>) for each unit– added to units.h

Steps to support nholthaus/units

- Need to declare/define "Value" class for attributes
 - Option 1: Define value class for each unit used in attributes (e.g., Watt, MilliWatt, etc.)
 - Option 2: Define value class for each dimension (e.g., Power)
 - fewer classes, but DoubleValue will only convert to one unit type
 - I have this option in my branch now, but may switch to or consider option 1
- Need to include namespaces of interest:

```
using units::power::dBm_t;
```

- and probably add aliases; e.g.

```
using Dbm = units::power::dBm_t;
```