ns-3 training

Tom Henderson

ns-3 annual meeting 2019

June 17-21, Florence, Italy

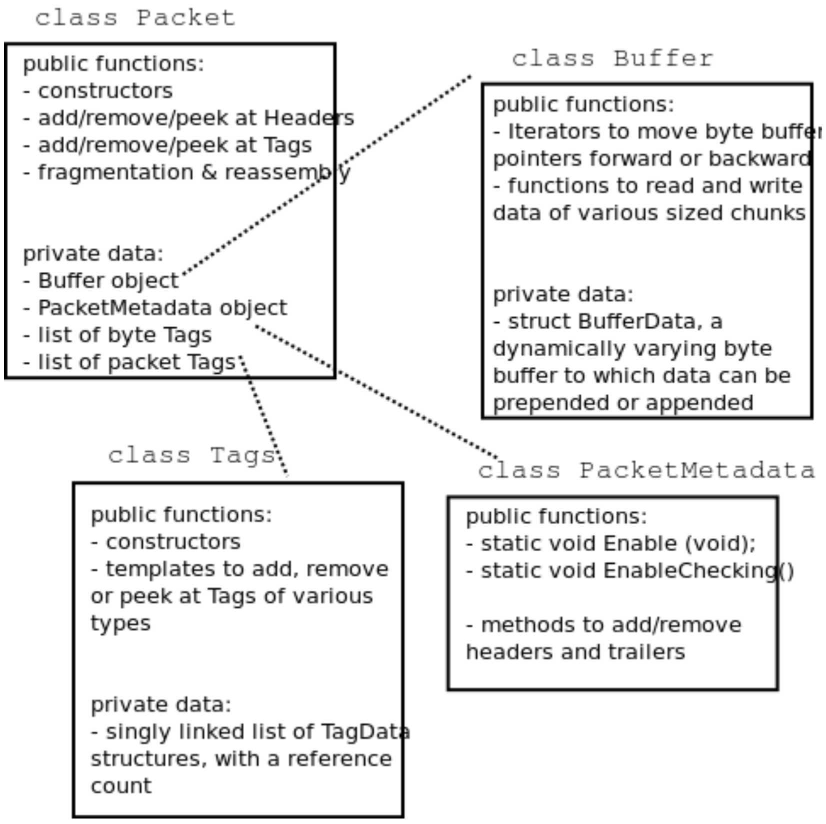
UNIVERSITY *of* WASHINGTON

ns-3 Packet

- Packet is an advanced data structure with the following capabilities
  - Supports fragmentation and reassembly
  - Supports real or virtual application data
  - Extensible
  - Serializable (for emulation)
  - Supports pretty-printing
  - Efficient (copy-on-write semantics)

ns-3 Packet structure

- Analogous to an mbuf/skbuff



Copy-on-write

- Copy data bytes only as needed

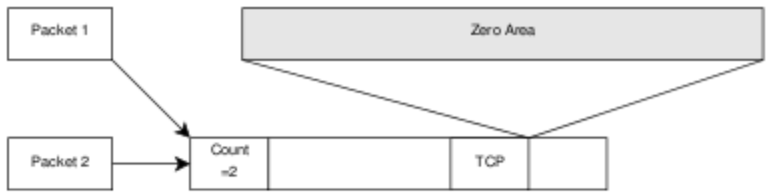


Figure 3.8: The TCP and the IP stacks hold references to a shared buffer.

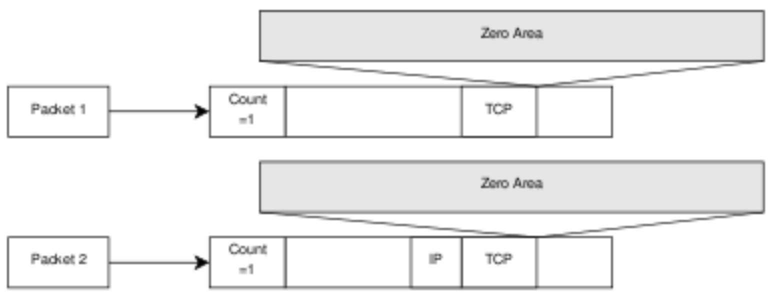


Figure 3.9: The IP stack inserts the IP header, triggers an un-share operation, completes the insertion.

Figure source: Mathieu Lacage's Ph.D. thesis

Headers and trailers

- Most operations on packet involve adding and removing an ns3::Header
- class ns3::Header must implement four methods:

```
Serialize()
Deserialize()
GetSerializedSize()
Print()
```

Headers and trailers (cont.)

- Headers are serialized into the packet byte buffer with Packet::AddHeader() and removed with Packet::RemoveHeader()
- Headers can also be 'Peeked' without removal

```
Ptr<Packet> pkt = Create<Packet> ();
UdpHeader hdr; // Note:  not heap allocated
pkt->AddHeader (hdr);
Ipv4Header iphdr;
pkt->AddHeader (iphdr);
```

Packet tags

- Packet tag objects allow packets to carry around simulator-specific metadata
  - Such as a "unique ID" for packets or cross-layer info
- Tags may associate with byte ranges of data, or with the whole packet
  - Distinction is important when packets are fragmented and reassembled
- Tags presently are not preserved across serialization boundaries (e.g. MPI)

PacketTag vs. ByteTag

- Two tag types are available: PacketTag and ByteTag
  - ByteTags run with bytes
  - PacketTags run with packets
- When Packet is fragmented, both copies of Packet get copies of PacketTags
- When two Packets are merged, only the PacketTags of the first are preserved
- PacketTags may be removed individually; ByteTags may be removed all at once

Tag example

- Here is a simple example illustrating the use of tags from the code in src/internet/model/udp-socket-impl.cc:

```
Ptr<Packet> p;  // pointer to a pre-existing packet
SocketIpTtlTag tag
tag.SetTtl (m_ipMulticastTtl); // Convey the TTL from
UDP layer to IP layer
p->AddPacketTag (tag);
```

- This tag is read at the IP layer, then stripped (src/internet/model/ipv4-l3-protocol.cc):

```
uint8_t ttl = m_defaultTtl;
SocketIpTtlTag tag;
bool found = packet->RemovePacketTag (tag);
if (found)
  {
    ttl = tag.GetTtl ();
  }
```

Packet metadata

- Packets may optionally carry metadata
  - record every operation on a packet's buffer
  - implementation of Packet::Print for pretty-printing of the packet
  - sanity check that when a Header is removed, the Header was actually present to begin with
- Not enabled by default, for performance reasons
- To enable, insert one or both statements:

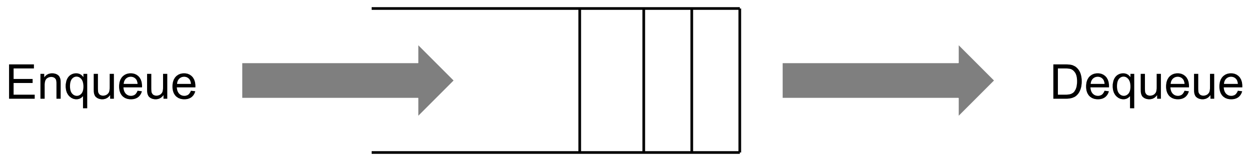
```
Packet::EnablePrinting ();
Packet::EnableChecking ();
```

Ptr<Packet>

- Packets are reference counted objects that support the smart pointer class `Ptr`
- Use a templated "Create" method instead of CreateObject for ns3::Objects
- Typical creation:
  - `Ptr<Packet> pkt = Create<Packet> ();`
- In model code, Packet pointers may be const or non-const; often Packet::Copy() is used to obtain non-const from const
  - `Ptr<const Packet> cpkt = ...;`
  - `Ptr<Packet> p = cpkt->Copy ();`

Queues in ns-3

- Queues are objects for storing packets

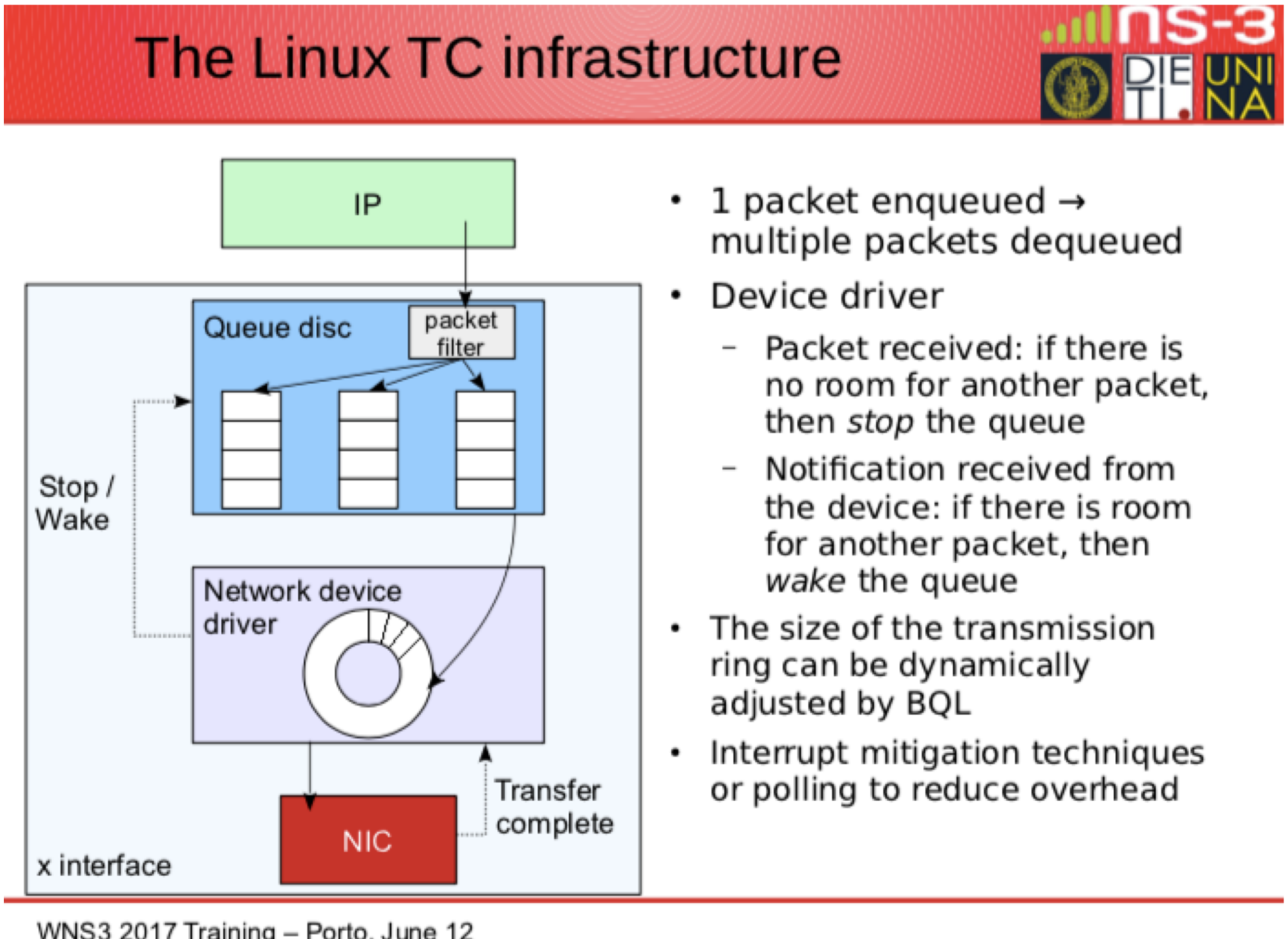


Common operations: GetNBytes (); GetNPackets (); etc.

- A templated Queue class exists to support a few use cases
  - simple queues such as a DropTail
  - WifiMacQueue
  - a Linux-like QueueDisc class

Linux-like TC architecture in ns-3

- Figure source: Stefano Avallone (2017 training)



Debugging support

- Assertions: NS_ASSERT (expression);
  - Aborts the program if expression evaluates to false
  - Includes source file name and line number
- Unconditional Breakpoints: NS_BREAKPOINT ();
  - Forces an unconditional breakpoint, compiled in
- Debug Logging (not to be confused with tracing!)
  - Purpose
    - Used to trace code execution logic
    - For debugging, not to extract results!
  - Properties
    - NS_LOG* macros work with C++ IO streams
    - E.g.: NS_LOG_UNCOND (”I have received ” << p->GetSize () << ” bytes”);
    - NS_LOG macros evaluate to nothing in optimized builds
    - When debugging is done, logging does not get in the way of execution performance

Debugging support (cont.)

- Logging levels:
  - NS_LOG_ERROR (...): serious error messages only
  - NS_LOG_WARN (...): warning messages
  - NS_LOG_DEBUG (...): rare ad-hoc debug messages
  - NS_LOG_INFO (...): informational messages (eg. banners)
  - NS_LOG_FUNCTION (...):function tracing
  - NS_LOG_PARAM (...): parameters to functions
  - NS_LOG_LOGIC (...): control flow tracing within functions
- Logging ”components”
  - Logging messages organized by components
  - Usually one component is one .cc source file
  - NS_LOG_COMPONENT_DEFINE ("OlsrAgent");
- Displaying log messages. Two ways:
  - Programatically:
    - LogComponentEnable("OlsrAgent", LOG_LEVEL_ALL);
  - From the environment:
    - NS_LOG="OlsrAgent" ./my-program

Running C++ programs through gdb

- The gdb debugger can be used directly on binaries in the build directory
- An easier way is to use a waf shortcut

```
./waf --command-template="gdb %s" --run <program-
  name>
```

Running C++ programs through valgrind

- valgrind memcheck can be used directly on binaries in the build directory
- An easier way is to use a waf shortcut

```
./waf --command-template="valgrind %s" --run
  <program-name>
```

- Note: disable GTK at configure time when running valgrind (to suppress spurious reports)
- `./waf configure --disable-gtk --enable-tests ...`

Testing

- ns-3 models need tests verifiable by others (often overlooked)
  - Onus is on the simulation project to validate and document results
  - Onus is also on the researcher to verify results
- ns-3 strategies:
  - regression tests
    - Aim for ***event-based*** rather than ***trace-based***
  - unit tests for verification
  - validation of models on testbeds where possible
  - reuse of code

Test framework

- ns-3-dev is checked nightly on multiple platforms
  - Linux gcc-4.x, i386 and x86_64, OS X, FreeBSD clang, and Cygwin (occasionally)
- `./test.py` will run regression tests

Walk through test code, test terminology (suite, case), and examples of how tests are run

Improving performance

- Debug vs optimized builds
  - ./waf -d debug configure
  - ./waf -d debug optimized
- Build ns-3 with static libraries
  - ./waf --enable-static
- Use different compilers (icc)
  - has been done in past, not regularly tested