DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

800 ANNI 1222·2022

UNIVERSITÀ DEGLI STUDI DI PADOVA

W&M

SIGNET

signet.dei.unipd.it

Output results and data analysis

UNWiS - Padova (Italy)

30th of January – 3rd of February 2023

Filippo Campagnaro, Roberto Francescon, Angela Soldà, Michele Zorzi

filippo.campagnaro@unipd.it

Analyzing the results

How to output relevant information in DESERT and analyze it: best-practices and how-tos

Current state

- DESERT does not currently have a uniform data output mechanism
- For many reasons: one is that simulations have requirements so diverse that it was best to leave to the user their definition
- So, along with the module structure, we can also define the data that are outputted and their format

In the script closing function

The best way to output results in desert is via the Tcl script ending function: here we see an example

```
proc finish {} {
   global ns opt outfile
   if ($opt(verbose)) {
     puts "Simulation summary"
     puts "-----------------------------------------------"
     puts "Total simulation time : [expr $opt(stoptime)-$opt(starttime)] s"
     puts "Packet size           : $opt(pktsize) byte(s)"
     puts "CBR period            : $opt(cbr_period) s"
    }
}
```

This function is usually called `finish`: in it, you can perform all the statistical calculations needed

The `finish` procedure

(of course, the `finish` procedure can be called whatever one desires)

In this function, we can perform all the computations we need to get all the measurements we want

```
proc finish {} {
   global ns opt outfile
   set sum_throughput  0
   for {set i 0} {$i < $opt(nn)} {incr i}  {
       for {set j 0} {$j < $opt(nn)} {incr j} {
           set part_throughput  [$cbr($i,$j) getthr]
       }
       set sum_throughput [expr $sum_throughput + $part_throughput]
   }
   set mean_throughput [expr ($sum_throughput/(($opt(nn))*($opt(nn)-1)))]
   puts "Mean Throughput  :     $mean_throughput "
}
```

`finish`: write to file

Instead of printing to *stdout*, we can write to a file the metrics gathered in the finish function

```
proc finish {} {
   global ns opt outfilename
   set outfile [open $outfilename w+]
   for {set i 0} {$i < $opt(nn)} {incr i}  {
   ...
       puts $outfile [format "%3d %3d" [clock seconds] [$mean_throughput]]
   ...
   }
   close $outfile
}
```

Outputting from C++

Sometimes it is needed to access data and variables that are not available in the Tcl script or, you need to resort to the logs

In these cases, it is important to have a consistent format of the output logs

```
void
UwModule::printOnLog(LogLevel log_level, std::string module,
      std::string message)
{
  ...
}
```

Having a function dedicated can help

Log function in C++

Let's see how we can implement it (example)

```
enum class LogLevel { ERROR = 0, INFO = 1, DEBUG = 2 };
void printOnLog(LogLevel log_level, std::string module, std::string message);
```

```
void
UwModem::printOnLog(LogLevel log_level, std::string module, std::string message)
{
    LogLevel actual_log_level = getLogLevel();
    if (actual_log_level >= log_level) {

      double timestamp = getEpoch();
      std::string ll_descriptor = "";
      log2string(log_level, ll_descriptor);

      outLog << std::setprecision(15) << left << ll_descriptor
             << "::[" << timestamp << "]::" << module
             << "(" << nodeID << "):" << message << std::endl;
      outLog.flush();
    }
}
```

- very useful to discriminate **log levels**
- important also to put a **timestamp** reference

The message is the main content of the log message

Parsing log files

Having a consistent format across the log messages outputted from C++ is fundamental to ease the parsing

```
MODULE::DEBUG::startTx::TRANSMITTING_PACKET
MODULE::DEBUG::endTx::END_TRANSMISSION
MODULE::DEBUG::configure::SETTING_OPERATION
MODULE::DEBUG::configure::SETTING_PERIOD
MODULE::ERROR::startRx::UNREADABLE_HEADER
...
```

Log level: `LogLevel log_level`

Log message: `std::string message`

Name pf the module outputting the log: `std::string module`

Then we can parse with whatever tool we are more comfortable with

Tracefiles

DESERT inherits from ns2 the *tracing* support

Tracing is way of recording reception, transmission and dropping of packets

```
set node($id) [$ns create-M_Node $opt(tracefile) $opt(cltracefile)]
```