Mobile and Wireless Networks Performance Analysis

- General guidelines to performance analysis
- Introduction to OMNET++
- Study case 1: A simple wired network
- Study case 2: A simple MANET
- Study case 3: A simple VANET
Often one needs to design and conduct an experiment in order to:

- demonstrate that a new technique or concept is feasible
- demonstrate that a new method is better than an existing method
- understand the impact of various factors and parameters on the overall system performance

There is a whole field of computer science called computer systems performance evaluation that does exactly this.

PERF EVAL 101: THE BASICS

- There are three main methods used in the design of performance studies:
  - **Experimental** approaches
    - measurement and use of a real system
  - **Simulation** approaches
    - design and use of computer simulations and simplified models to assess performance
  - **Analytic** approaches
    - the use of mathematics, Markov chains, queueing theory, Petri Nets, abstract models...
Three Rules of Validation

- Do not trust the results of a measurement until they have been validated by simulation or analytical modeling.

- Do not trust the results of a simulation model until they have been validated by analytical modeling or measurements.

- Do not trust the results of an analytical model until they have been validated by a simulation model or measurements.
EXPERIMENTAL DESIGN AND METHODOLOGY

- The design of a performance study requires great care in experimental design and methodology.
- Need to identify:
  1. experimental factors to be tested
  2. levels (settings) for these factors
  3. performance metrics to be used
  4. experimental design to be used
FACTORS

- Factors are the main “components” or “things” that are to be varied in an experiment, because their impact on performance wants to be understood.
- Examples: offered load, switch size, number of buffers at output ports.
- Need to choose factors properly, since the number of factors affects size of study.
LEVELS

- Levels are the precise settings of the factors that are to be used in an experiment.
- Examples: switch size $N = 2, 4, 8, 16$
- Example: buffer size $B = 100, 200, 400, 800$
- Need to choose levels realistically
- Need to cover reasonable portion of the design space
PERFORMANCE METRICS

- Performance metrics specify what you want to measure in your performance study.
- Examples: packet loss, inter-packet delay.
- Must choose your metrics properly and instrument your experiment accordingly.
Case Study: Two Congestion Control Algorithms

- **Service:** Send packets from specified source to specified destination in order.

- **Possible outcomes:**
  - Some packets are delivered in order to the correct destination.
  - Some packets are delivered out-of-order to the destination.
  - Some packets are delivered more than once (duplicates).
  - Some packets are dropped on the way (lost packets).
Case Study (Cont)

○ Performance: For packets delivered in order,
  ○ Time-rate-resource ⇒
    ○ Response time to deliver the packets
  ○ Throughput: the number of packets per unit of time.
  ○ Processor time per packet on the source end system.
  ○ Processor time per packet on the destination end systems.
  ○ Processor time per packet on the intermediate systems.

○ Variability of the response time ⇒ Retransmissions
  ○ Response time: the delay inside the network
Case Study (Cont)

- Out-of-order packets consume buffers ⇒ Probability of out-of-order arrivals.
- Duplicate packets consume the network resources ⇒ Probability of duplicate packets
- Lost packets require retransmission ⇒ Probability of lost packets
- Too much loss cause disconnection ⇒ Probability of disconnect
Case Study (Cont)

- Throughput and delay were found redundant
  ➔ Use Power.

- \[ \text{Power} = \frac{\text{Throughput}}{\text{Response Time}} \]
Capacity

Throughput

Knee

Nominal capacity

Knee capacity

Usable capacity

Load

Response time

Load
Nominal Capacity: Maximum achievable throughput under ideal workload conditions. E.g., bandwidth in bits per second. The response time at maximum throughput is too high.

Usable capacity: Maximum throughput achievable without exceeding a pre-specified response-time limit

Knee Capacity: Knee = Low response time and High throughput
Common Performance Metrics (Cont)

- **Efficiency**: Ratio usable capacity to nominal capacity. Or, the ratio of the performance of an n-processor system to that of a one-processor system is its efficiency.
- **Utilization**: The fraction of time the resource is busy servicing requests. Average fraction used for memory.

![Graph showing efficiency vs. number of processors]
Common Performance Metrics (Cont)

- **Reliability:**
  - Probability of errors
  - Mean time between errors (error-free seconds).

- **Availability:**
  - Mean Time to Failure (MTTF)
  - Mean Time to Repair (MTTR)
  - \( \frac{MTTF}{MTTF+MTTR} \)
Experimental design refers to the organizational structure of your experiment

Need to methodically go through factors and levels to get the full range of experimental results desired

There are several “classical” approaches to experimental design

- One factor at a time
  - vary only one factor through its levels to see what the impact is on performance
- Two factors at a time
  - vary two factors to see not only their individual effects, but also their interaction effects, if any
- Full factorial
  - try every possible combination of factors and levels to see full range of performance results
OTHER ISSUES

- Simulation run length
  - choosing a long enough run time to get statistically meaningful results (equilibrium)

- Simulation start-up effects and end effects
  - deciding how much to “chop off” at the start and end of simulations to get proper results

- Replications
  - ensure repeatability of results, and gain greater statistical confidence in the results given

- Presentation of results
Guidelines for Preparing Good Charts

1. Require minimum effort from the reader
   - Direct labeling vs. legend box

2. Maximize Information: Words in place of symbols
   - Clearly label the axes
Guidelines (cont)

- Minimize Ink: No grid lines, more details

- Use Commonly accepted practices: origin at (0,0) Independent variable (cause) along x axis, linear scales, increasing scales, equal divisions

- Avoid ambiguity: Show coordinate axes, scale divisions, origin. Identify individual curves and bars.
Common Mistakes in Preparing Charts

- Presenting too many alternatives on a single chart
  Max 5 to 7 messages => Max 6 curves in a line chart, no more than 10 bars in a bar chart, max 8 components in a pie chart
- Presenting many y variables on a single chart
Common Mistakes in Charts (Cont)

- Using symbols in place of text

![Graphs showing symbols and key words]

- Placing extraneous information on the chart: grid lines, granularity of the grid lines
- Selecting scale ranges improperly: automatic selection by programs may not be appropriate
Using a line chart in place of column chart: line => Continuity
Pictorial Games

- Using non-zero origins to emphasize the difference
  Three quarter high-rule => height/width > 3/4
Pictorial Games (Cont)

- Using double-whammy graph for dramatization
  Using related metrics
Pictorial Games (Cont)

- Plotting random quantities without showing confidence intervals
Using inappropriate cell size in histograms

- [0,2)
- [2,4)
- [4,6)
- [6,8)
- [8,10)
- [10,12)

- [0,6)
- [6,12)

Response Time

Frequency
Reasons for not Accepting an Analysis

- This needs more analysis.
- You need a better understanding of the workload.
- It improves performance only for long IOs/packets/jobs/files, and most of the IOs/packets/jobs/files are short.
- It improves performance only for short IOs/packets/jobs/files, but who cares for the performance of short IOs/packets/jobs/files, its the long ones that impact the system.
- It needs too much memory/CPU/bandwidth and memory/CPU/bandwidth isn't free.
- It only saves us memory/CPU/bandwidth and memory/CPU/bandwidth is cheap.
Mobile and Wireless Networks
Performance Analysis

- General guidelines to performance analysis
- Introduction to OMNET++
- Study case 1: A simple wired network
- Study case 2: A simple MANET
- Study case 3: A simple VANET
What is OMNeT++?

- OMNeT++ is a component-based, modular and open-architecture discrete event network simulator.
- OMNeT++ represents a framework approach
  - Instead of containing explicit and hardwired support for computer networks or other areas, it provides an infrastructure for writing such simulations
  - Specific application areas are catered by various simulation models and frameworks, most of them open source.
  - These models are developed completely independently of OMNeT++, and follow their own release cycles.
OMNET++ Frameworks

- Partial list of OMNeT++-based network simulators and simulation frameworks:
  - Mobility Framework -- for mobile and wireless simulations
  - INET Framework -- for wired and wireless TCP/IP based simulations
  - Castalia -- for wireless sensor networks
  - MiXiM -- for mobile and wireless simulations

- More specialized, OMNeT++-based simulators:
  - OverSim -- for overlay and peer-to-peer networks (INET-based)
  - NesCT -- for TinyOS simulations
  - Consensus Positif and MAC Simulator -- for sensor networks
  - SimSANs -- for storage area networks
  - CDNSim -- for content distribution networks
  - ACID SimTools -- for simulation of concurrency control, atomic commit processing and recovery protocols
  - X-Simulator -- for testing synchronization protocols
  - FIELDDBUS -- for simulation of control networks (fieldbuses)
  - PAWiS -- Power Aware Wireless Sensor Networks Simulation Framework
Important issues in a discrete event simulation environment

- Pseudorandom generators
- Flexibility
- Programming model
- Model management
- Support for hierarchical models
- Debugging, tracing, and experiment specifications
- Documentation
- Large scale simulation
- Parallel simulation
OMNET++ Programming model

- Simulated objects are represented by modules
  - Modules can be simple or composed (depth of module nesting is not limited)
  - Modules communicate by messages (sent directly or via gates)
  - One module description consists of:
    - Interface description (.NED file)
    - Behavior description (C++ class)

- Modules, gates and links can be created:
  - Statically - at the beginning of the simulation (NED file)
  - Dynamically – during the simulation
Network interface card, a compound module consisting of a simple module MAC and a compound module Phy.
Why use separate NED and ini files?

- NED files define the topology of network/modules
  - It is part of the model description

- Ini files define
  - Simulation parameters
  - Results to collect
  - Random seeds

- This separation allows to change parameters without modifying the model
  - E.g. no need to recompile, experiments can be executed as a batch
Running a model (omnetpp.ini)

[General]
  network = etherLAN
  *.numStations = 20
  **.frameLength = normal(200,1400)
  **.station[0].numFramesToSend = 5000
  **.station[1-5].numFramesToSend = 1000
  **.station[*].numFramesToSend = 0

One function of the ini file is to tell which network to simulate. You can also specify in there which NED files to load dynamically, assign module parameters, specify how long the simulation should run, what seeds to use for random number generation, how much results to collect, set up several experiments with different parameter settings, etc.
Debugging and tracking

- Support is offered for:
  - Recording data vectors and scalars in output files
  - Random numbers (also from several distributions) with different starting seeds
  - Tracing and debugging aids (displaying info about the module’s activity, snapshots, breakpoints)

- Simulations are easy to configure using .ini file

- Batch execution of the same simulation for different parameters is also included

- Simulations may be run in two modes:
  - Command line: Minimum I/O, high performance.
  - Interactive GUI: Tcl/Tk windowing, allows view what’s happening and modify parameters at run-time.
Output of a simulation

- The simulation may write output vector and output scalar files
  - omnetpp.vec and omnetpp.sca
  - The capability to record simulation results has to be explicitly programmed into the simple modules
- An output vector file contains several output vectors
  - series of pairs timestamp, value
  - They can store things like:
    - queue length over time, end-to-end delay of received packets, packet drops or channel throughput
  - You can configure output vectors from omnetpp.ini
    - you can enable or disable recording individual output vectors, or limit recording to a certain simulation time interval
- Output vectors capture behaviour over time
- Output scalar files contain summary statistics
  - number of packets sent, number of packet drops, average end-to-end delay of received packets, peak throughput
Links and readings

- “The OMNeT++ discrete event simulation system”, Varga A. Proceedings of the European Simulation Multiconference (ESM 2001), Prague, Czech Republic, 6–9 June 2001
- “Comparison of OMNET++ and other simulator for WSN simulation”, Xiaodong Xian; Weiren Shi; He Huang Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on
- “Simulating wireless and mobile networks in OMNeT++ the MiXiM vision”, Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems Marseille, France, 2008