FlowMonitor — a network monitoring framework for the Network Simulator 3 (NS-3)

Gustavo Carneiro Pedro Fortuna Manuel Ricardo

INESC Porto — Unidade de Telecomunicações e Multimédia
Faculdade de Engenharia da Universidade do Porto

2009-10-19 / NSTools 2009

Outline

1. **Introduction**
   - Network Monitoring
   - Simulation Overview
   - NS-3
2. **The FlowMonitor NS-3 module**
   - Requirements
   - Architecture
   - Flow Data Structures
     - Basic Metrics
3. **Example**
4. **Validation and Results**
   - Validation
   - Performance Results
5. **Conclusions**
   - Conclusions
   - Future Work

Outline

1. **Introduction**
   - Network Monitoring
   - Simulation Overview
   - NS-3
2. **The FlowMonitor NS-3 module**
   - Requirements
   - Architecture
   - Flow Data Structures
     - Basic Metrics
3. **Example**
4. **Validation and Results**
   - Validation
   - Performance Results
5. **Conclusions**
   - Conclusions
   - Future Work

Network Monitoring

- Traditional network monitoring issues:

strategy: passive (just measures) or active (injects traffic to be measured)
- In simulation, the researcher is already injecting flows, we just need to measure them

monitoring points: not every network element can be easily monitored (lack of SNMP)
- In simulation, every node is monitorable

monitoring duration: must be large enough to enable the gathering of statistically sound results
- In simulation, this can be easily controlled

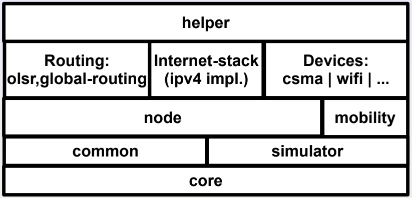
synchronization: events to monitor must be ordered among a set of nodes
- Usually a time stamp in messages is required
- Problem: synchronizing the clocks
- In simulation, the clocks are always synchronized

transparency: monitoring control traffic affects results
- In simulation, monitoring data is reported via method calls, not packets

Simulation Overview

- Writing simulations is a complex task
  - Write a model
  - Define a topology
  - Add flows
  - Debug...
  - Measure the flows (tracing plus pre-processing)
  - Write the flow statistics to a file for post-processing
- Significant portion of time dedicated to write code to measure flows and collect statistics
- Solution: “Flow Monitor” for NS-3
  - Just a few lines of code
  - Measures all flows in the simulation, no questions asked
  - Provides flow statistics as
    1. Simple in-memory data structures
    2. Serialized to an XML file

NS-3



core base runtime system: attributes, tracing, smart pointers, callbacks

common packets, buffers, and packet headers

simulator event scheduler, high precision time class

node network simulator fundamentals: Node, NetDevice, addresses

mobility mobility models (describe the trajectory of a node)

olsr the OLSR routing protocol (adhoc)

global-routing “GOD” routing

internet-stack UDP/TCP/IPv4/6 implementation

csma, wifi, etc. actual implementations of the NetDevice class

helper a “façade” that keeps scripting nice and neat

Outline

1. Introduction
   - Network Monitoring
   - Simulation Overview
   - NS-3
2. The FlowMonitor NS-3 module
   - Requirements
   - Architecture
   - Flow Data Structures
     - Basic Metrics
3. Example
4. Validation and Results
   - Validation
   - Performance Results
5. Conclusions
   - Conclusions
   - Future Work

ease of use: It must be easy to use

- Activate with just a few lines of code
- Detect passing flows automatically
- Little or no configuration required

amount of data to capture: Balanced amount of data

- Too much captured data:
  - Incurs disk I/O bandwidth
  - Slows down simulations
  - Makes data transfer and manipulation painfully slow
- Too little and we may miss something important
  - Often we do not realize it is important until too late

support python: “fire-and-forget” approach, no callbacks

- NS-3’s Config::Connect uses callbacks to process per-packet events
- In Python, per-packet processing is unpractical, way too slow

output data format: must be easy to process

- Candidates:

  binary files (e.g. HDF): disk efficient, fast, difficult to read, difficult to extend

  ASCII traces: verbose, difficult to extend, potentially slow to read

  XML files: verbose, easy to extend, slow but easy to read

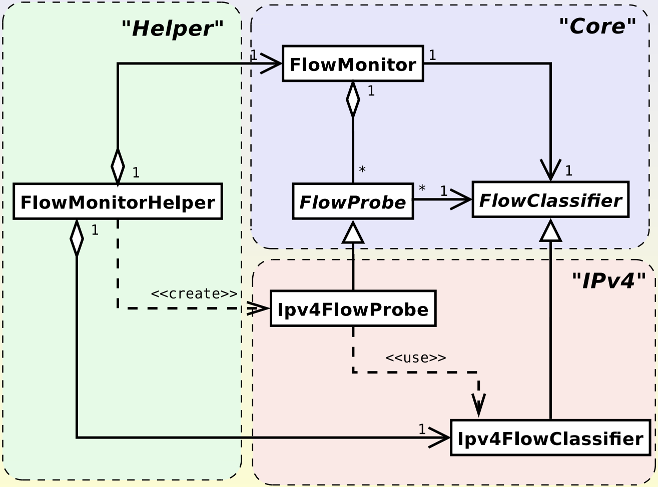
  SQL database: efficient and fast, difficult to read (table format is hidden away), difficult to extend
- Chosen method: XML
  - Runner up (future work): SQL database
  - Besides, in-memory data structures always available!

extensible: allow extension to monitor different kinds of flows

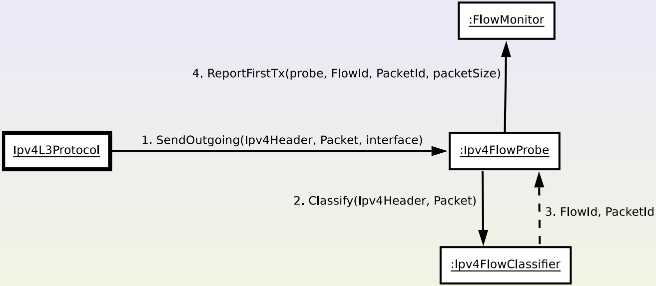
- The concept of “flow” is not set in stone
- We *usually* mean TCP-or-UDP L4 flows
  - The five-tuple filter: (source IP, destination IP, protocol, source port, destination port)
- Sometimes we want other kinds of flows, e.g.:
  - L2 flows: (source mac, destination mac, ethertype)
- The way to detect flows may vary
  - Overlay networks
  - Tunnels

minimize overhead: monitoring adds time/memory overhead

Architecture

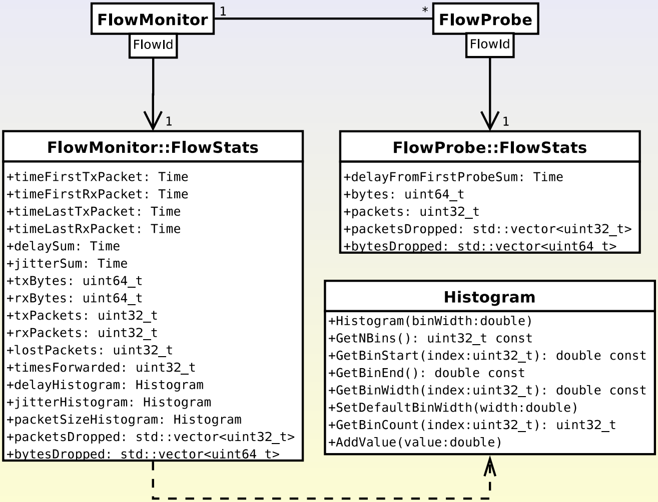


Architecture

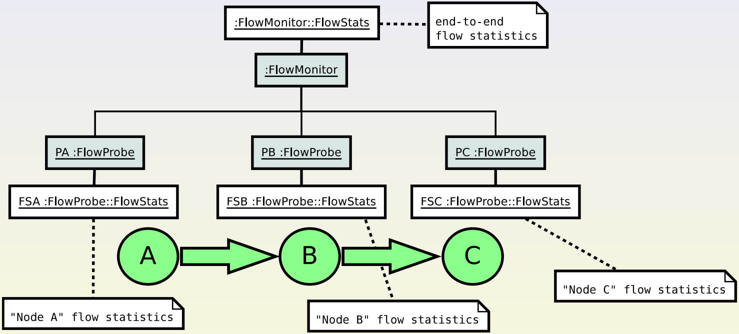


- FlowId: 32-bit identifier of a flow
- PacketId: 32-bit identifier of a packet within a flow
- The class FlowMonitor remains **unaware** of whether it is monitoring IPv4, IPv6, or MAC, flows
  - Only abstract identifiers are used, not packet headers
  - Most of the flow monitoring design, data structures, and code is **reused** when extending to monitor different kinds of flows

Flow Data Structures



Flow Data Structures



- Per-probe flow statistics can answer:
  - Which hop accounts for most of the packet losses?
  - Which hop accounts for most of the end-to-end delay?

timeFirstTxPacket, timeLastTxPacket begin and end times of the flow from the point of view of the receiver

timeFirstRxPacket, timeLastRxPacket begin and end times of the flow from the point of view of the receiver

delaySum, jitterSum sum of delay and jitter values

txBytes, txPackets number of transmitted bytes and packets

rxBytes, rxPackets number of received bytes and packets

lostPackets number of definitely lost packets

timesForwarded the number of times a packet has been reportedly forwarded, summed for all packets in the flow

Flow Data Structures

delayHistogram, jitterHistogram, packetSizeHistogram
: Histogram versions for the delay, jitter, and packet sizes, respectively

packetsDropped, bytesDropped discriminates the losses by a *reason code*

- **DROP_NO_ROUTE** no IPv4 route found for a packet
- **DROP_TTL_EXPIRE** a packet was dropped due to an IPv4 TTL field decremented and reaching zero
- **DROP_BAD_CHECKSUM** a packet had a bad IPv4 header checksum and had to be dropped

Other metrics can be derived from the basic metrics:

mean delay: $\overline{delay} = \frac{delaySum}{rxPackets}$

mean jitter: $\overline{jitter} = \frac{jitterSum}{rxPackets-1}$

mean transmitted packet size (byte): $\overline{S_{tx}} = \frac{txBytes}{txPackets}$

mean received packet size (byte): $\overline{S_{rx}} = \frac{rxBytes}{rxPackets}$

mean transmitted bitrate (bit/s):

$$\overline{B_{tx}} = \frac{8 \cdot txBytes}{timeLastTxPacket - timeFirstTxPacket}$$

mean received bitrate (bit/s):

$$\overline{B_{rx}} = \frac{8 \cdot rxBytes}{timeLastRxPacket - timeFirstRxPacket}$$

mean hop count: $\overline{hopcount} = 1 + \frac{timesForwarded}{rxPackets}$

packet loss ratio: $q = \frac{lostPackets}{rxPackets+lostPackets}$

Outline

1. Introduction
   - Network Monitoring
   - Simulation Overview
   - NS-3
2. The FlowMonitor NS-3 module
   - Requirements
   - Architecture
   - Flow Data Structures
     - Basic Metrics
3. Example
4. Validation and Results
   - Validation
   - Performance Results
5. Conclusions
   - Conclusions
   - Future Work

```
flowmon_helper = ns3.FlowMonitorHelper()
monitor = flowmon_helper.InstallAll()
monitor.SetAttribute("DelayBinWidth",
                     ns3.DoubleValue(0.001))
monitor.SetAttribute("JitterBinWidth",
                     ns3.DoubleValue(0.001))
monitor.SetAttribute("PacketSizeBinWidth",
                     ns3.DoubleValue(20))

ns3.Simulator.Run()

monitor.SerializeToXmlFile("results.xml", True, True)
```

1. Create a new FlowMonitorHelper object;
2. Call the method InstallAll on this object
   - FlowMonitor is created
   - IPv4 probes are installed in all nodes
3. Configure some histogram attributes
4. Run the simulation, as before, calling ns3.Simulator.Run()
5. Finally, write the flow monitored results to a XML file

Outline

1 Introduction
- Network Monitoring
- Simulation Overview
- NS-3

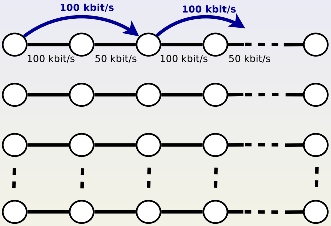
2 The FlowMonitor NS-3 module
- Requirements
- Architecture
- Flow Data Structures
  - Basic Metrics

3 Example

4 Validation and Results
- Validation
- Performance Results

5 Conclusions
- Conclusions
- Future Work

Validation



- Point-to-point links connecting the nodes
- Link capacities alternating between 100kbit/s and 50kbit/s
- Maximum queue size: 100 packets
- Every other nodes sends a 100 kbit/s flow to node two hops away
- Half the packets expected to be lost

Validation

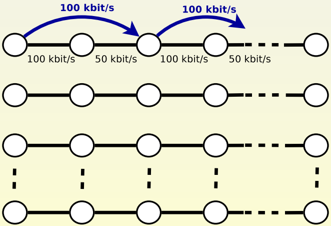
$$
\begin{aligned}
C_1 &= 100000 \text{ bit/s} \\
S &= (512 + 20 + 8 + 2) \times 8 = 4336 \text{ bit} \\
d_1 &= \frac{S}{C_1} = 0.04336 \text{ s} \\
C_2 &= 50000 \text{ bit/s} \\
d_2 &= 101 \times \frac{S}{C_2} = 8.75872 \text{ s} \\
d_{total} &= d_1 + d_2 = 8.80208 \text{ s} \\
B_{tx} &= \frac{512 + 20 + 8}{512 + 20 + 8 + 2} \times C_1 = 99631.00 \text{ bit/s} \\
B_{rx} &= \frac{512 + 20 + 8}{512 + 20 + 8 + 2} \times C_2 = 49815.50 \text{ bit/s}
\end{aligned}
$$

Metric	Measured Value (95% C. I.)	Expected Value	Mean Error
Tx. bitrate	$99646.06 \pm 2.68 \times 10^{-5}$	99631.00	+0.015 %
Rx. bitrate	$49832.11 \pm 7.83 \times 10^{-5}$	49815.50	+0.033 %
Delays	$8.8005 \pm 8.8 \times 10^{-9}$	8.80208	-0.018 %
Losses	$0.4978 \pm 1.5 \times 10^{-6}$	0.5000	-0.44 %

Table: Validation results

Performance Results

- Run series of simulations, validation scenario
  - Increasing the network size between 16 and 1024 nodes
  - Measure time and memory taken to simulate scenario
  - Compare three variants:
    1. Without collecting any results
    2. With flow monitoring
    3. With ascii tracing to a file



Performance Results

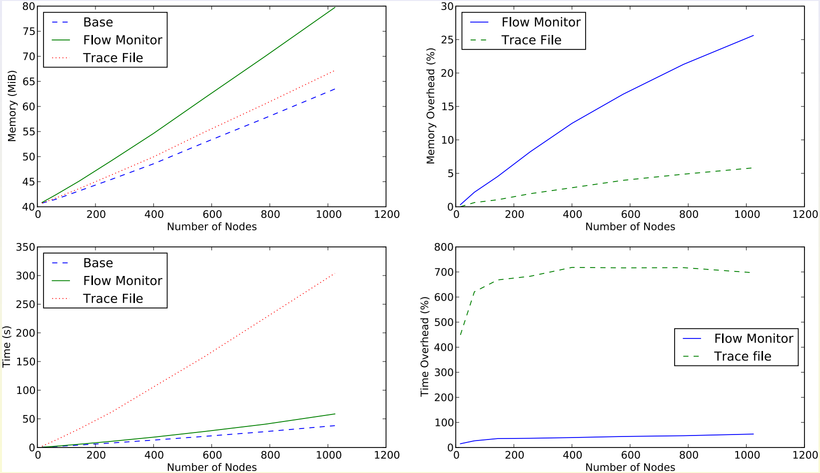


Figure: Performance results of the flow monitor

Outline

1. Introduction
   - Network Monitoring
   - Simulation Overview
   - NS-3
2. The FlowMonitor NS-3 module
   - Requirements
   - Architecture
   - Flow Data Structures
     - Basic Metrics
3. Example
4. Validation and Results
   - Validation
   - Performance Results
5. Conclusions
   - Conclusions
   - Future Work

Conclusions

- A common problem was identified
  - "how to easily extract flow metrics from arbitrary simulations?"
- Existing solutions do not solve this problem effectively
- The **Flow Monitor** solves the problem
  - Requires significant less programming time than NS-3 callback based tracing
  - A lot more efficient than ascii tracing

Future Work

- More data output methods (e.g. database and binary file)
- More options to control level of detail stored in memory
- Monitor multicast/broadcast flows
- Closer integration with NetDevices,
  - Monitor packet drop from NetDevice's transmission queue
  - Handle transmission errors from layer 2
- Record how flow metrics evolve over time
  - By saving a periodic snapshot of the flows metrics to a file
- Add convenience methods to the Histogram class to compute the values
  - N (number of samples)
  - μ (mean)
  - s (standard error)