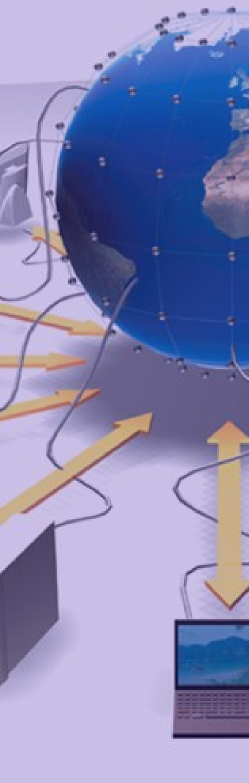


NEW TECHNOLOGIES FOR APPLICATION LEVEL MONITORING ON THE GRID

Maciej Woś, IT/GS

CERN IT Department
CH-1211 Genève 23
Switzerland

CERN

THE OUTLINE

1. The goal, motivation and context
2. Technology introduction:
   ActiveMQ, STOMP, OpenWire, Ganga, Diane
3. Implementation: producers, consumers and the most common monitoring scenario
4. Real world applications: Lattice QCD
5. Testing in the production environments
   (HammerCloud @ STEP09, LQCD, Geant4)
6. Synthetic testing
7. Conclusions

MOTIVATION AND CONTEXT

Dashboard Monitoring

- Job submission tools
  - For LHC experiments
    - Ganga/Diane (ATLAS / LHCb)
    - CRAB (CMS)
    - Panda (ATLAS)
  - For generic EGEE applications
- Data Management Systems (ATLAS)

Motivation

- Replace MonALISA transport layer (Dashboard) with MSG

THE GOAL OF MY WORK

Evaluation of WLCG Messaging System for Grids (MSG)

- Tests in the production environments
  - HammerCloud (STEP09)
  - LatticeQCD 2
  - Geant 4 regression testing (June 09)
- Synthetic tests of performance / scalability

ActiveMQ

- Open source (Apache 2.0 licensed) message broker which fully implements the Java Message Service 1.1 (JMS)
- Support for standard wire level protocols, including STOMP and Apache's own OpenWire

TECHNOLOGY INTRODUCTION (2)

OpenWire

- Binary protocol designed to marshal objects to byte arrays and back
- The Java OpenWire transport is the default transport in ActiveMQ 4.x or later

STOMP

- Streaming Text Orientated Messaging Protocol
- Makes it easy to write a client in pure Ruby, Perl, Python or PHP for working with ActiveMQ

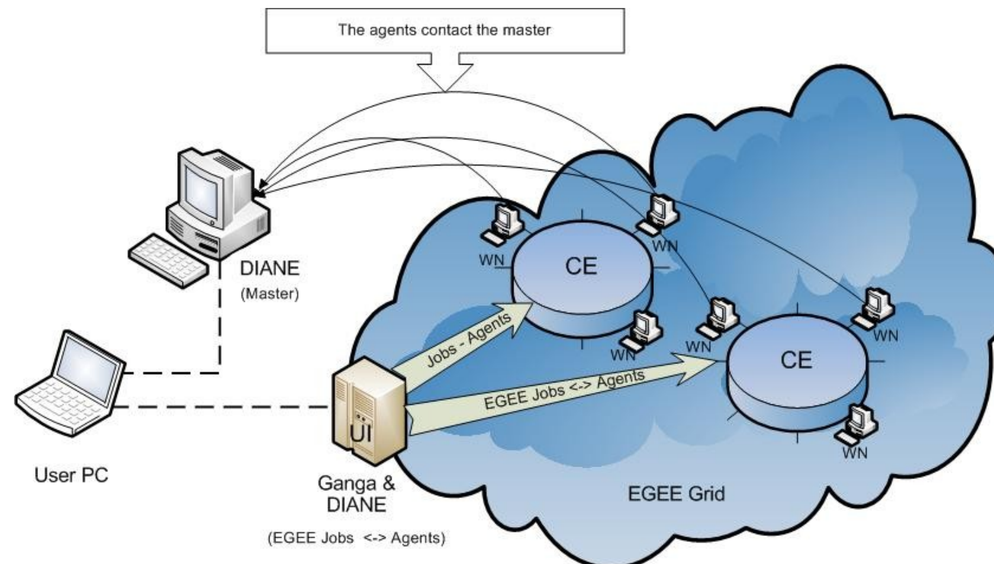
TECHNOLOGY INTRODUCTION (3)

Ganga

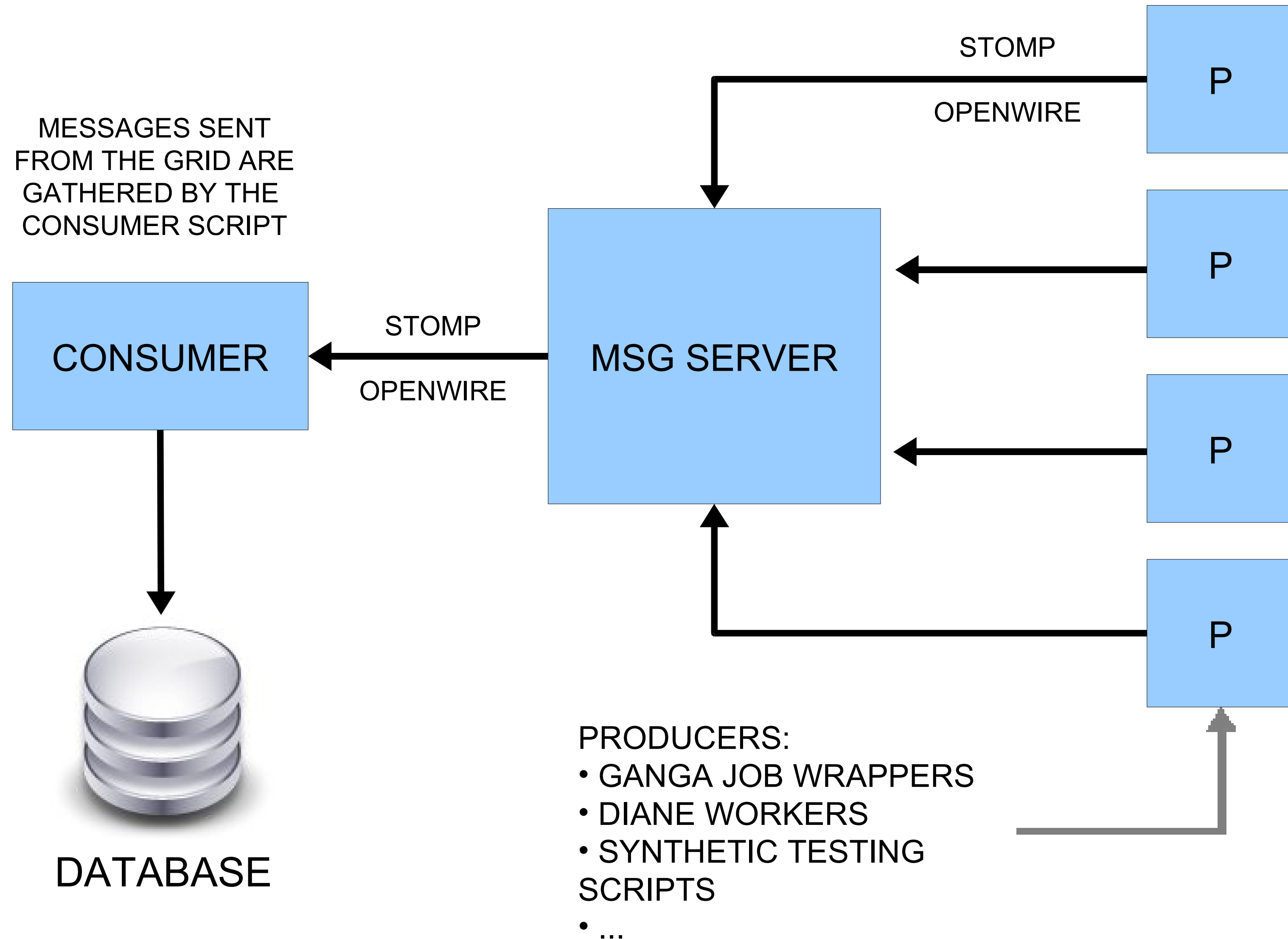
- Easy to use frontend for job definition and management, implemented in Python
- Main users at CERN: Atlas and LHCb

Diane

- Private, pilot based system
- Provides user- or application-specific Grid overlays for handling workloads in Master/Worker paradigm



THE MOST COMMON DASHBOARD MONITORING SCENARIO



THE CHOICE OF PROTOCOLS

- Ganga/Diane, Panda, Crab, ...
  - Pure Python, STOMP based implementation
  - Portable, easy to deploy
  - No connection management
  - No flow control

- Alternative: ActiveMQ native library
  - Suitable for Java / C / C++ / C#
  - OpenWire (better performance) instead of STOMP
  - Advanced features at the cost of portability

INTEGRATING MSG MONITORING WITH GANGA / DIANE

Monitoring in Ganga client and job wrappers

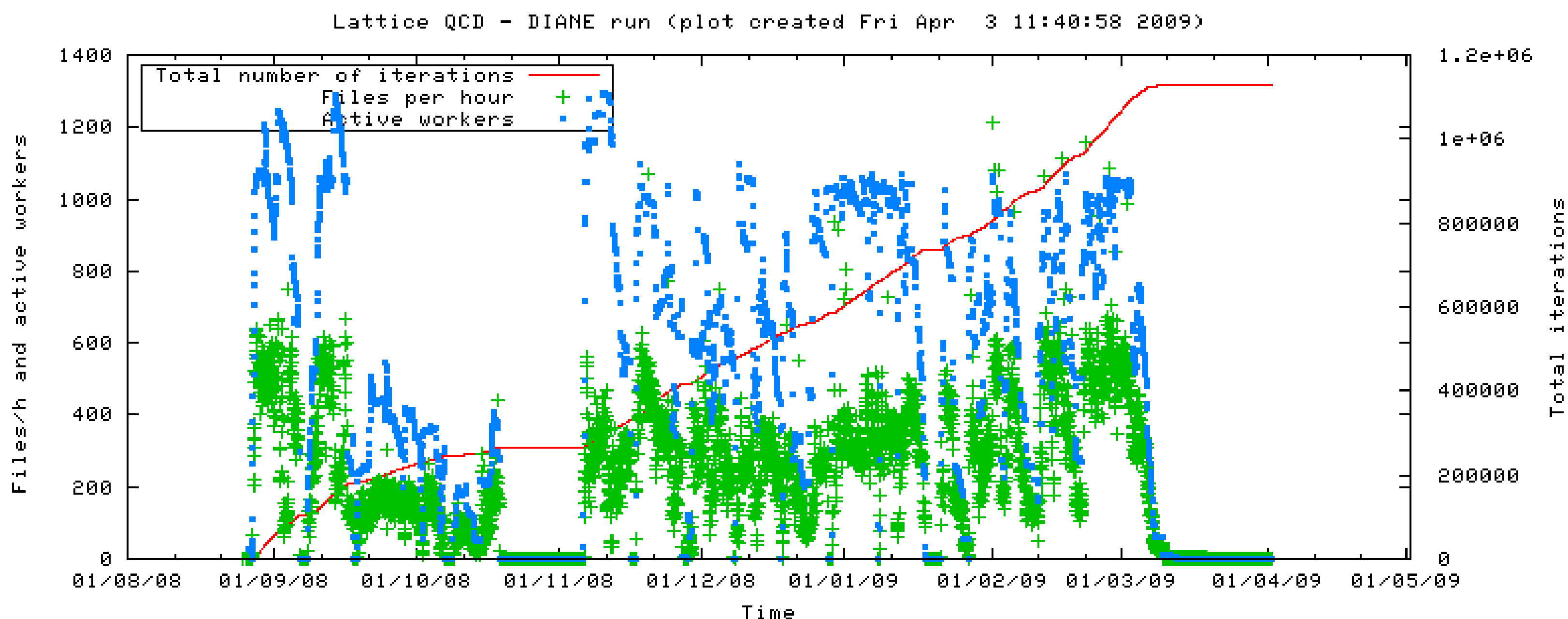
- MonitoringServices plugin based on existing Ganga monitoring framework

Monitoring in Diane master and workers

- Shares MSG handling code with Ganga, minimal changes to Diane codebase

EXAMPLE: LATTICE QCD

- No monitoring solutions for generic (non-LHC) activities
- Ad hoc monitoring methods:
  Python, Wiki, gnuplot, cron, scp, ...



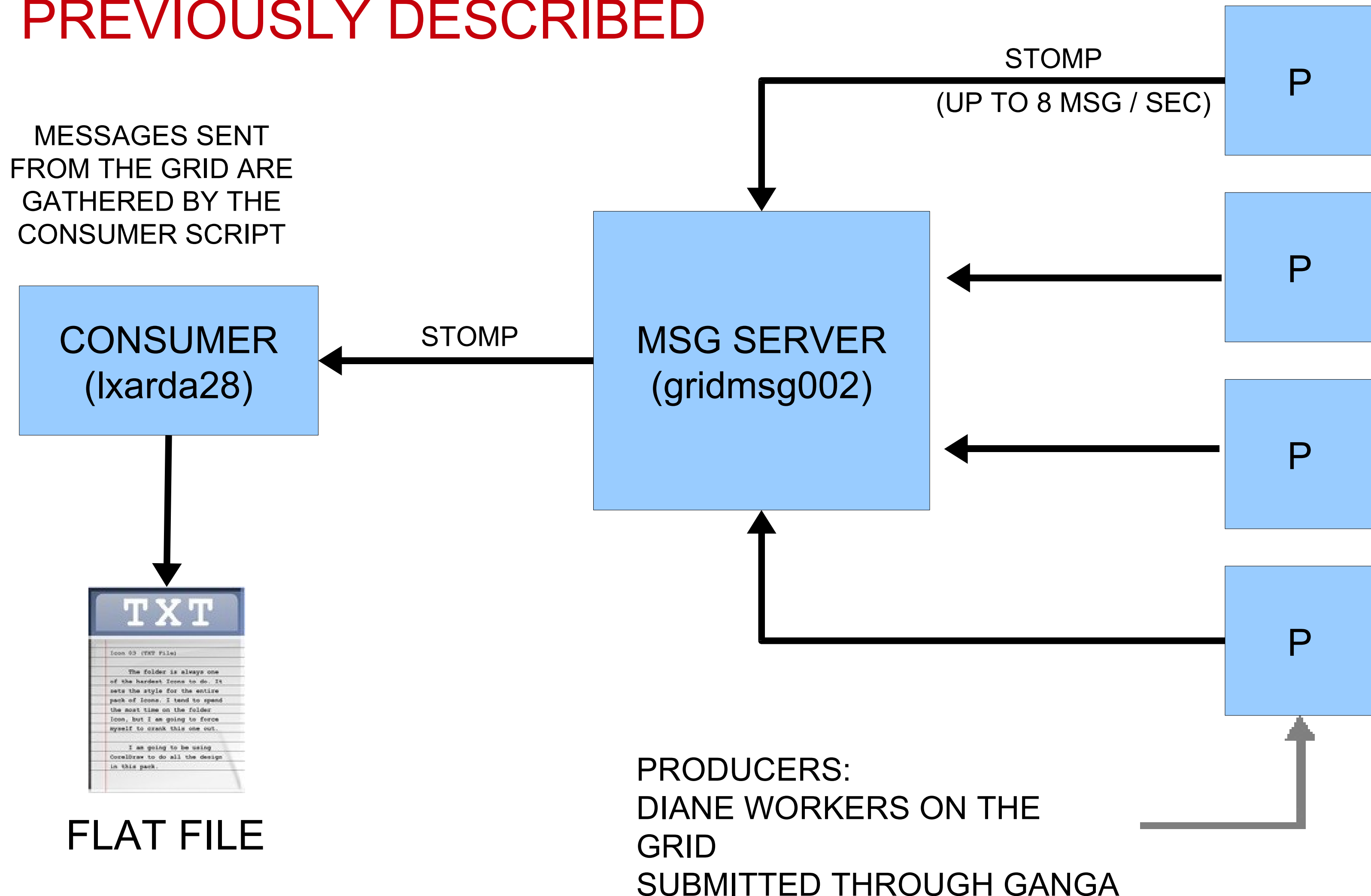
TESTS IN PRODUCTION ENVIRONMENTS

- LatticeQCD
  - 100 workers running over extnended period of time
- HammerCloud STEP09
  - Last days of STEP09, 10000 jobs during two days of execution
- Geant4 regression testing
  - 500 grid jobs – Diane workers submitted through Ganga

No issues with MSG monitoring were observed. Enabling monitoring plugins in Ganga / Diane does not influence the running application in any way.

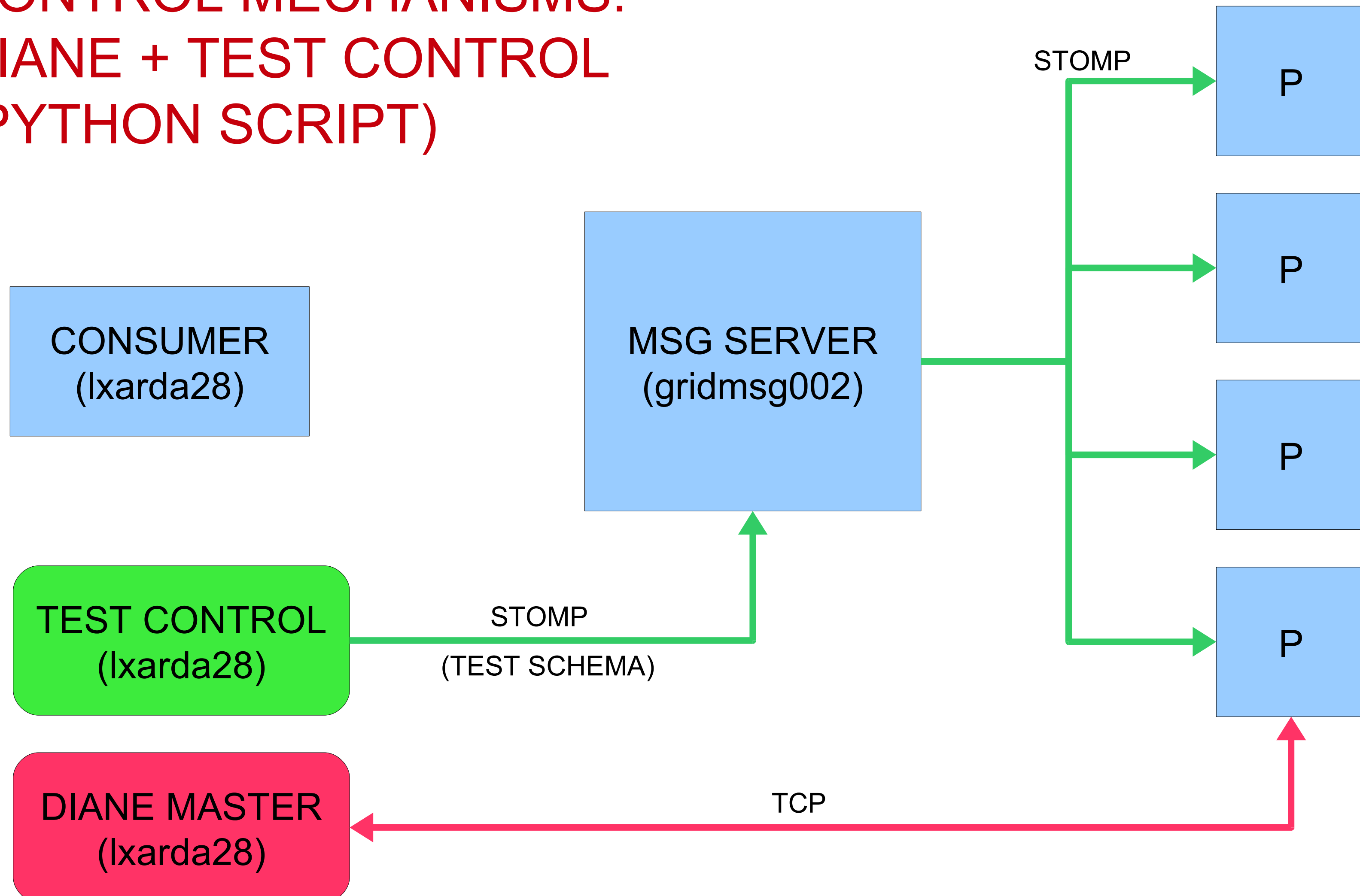
SYNTHETIC TESTING

THE SAME MESSAGE MODEL AS PREVIOUSLY DESCRIBED



SYNTHETIC TESTING

CONTROL MECHANISMS: DIANE + TEST CONTROL (PYTHON SCRIPT)

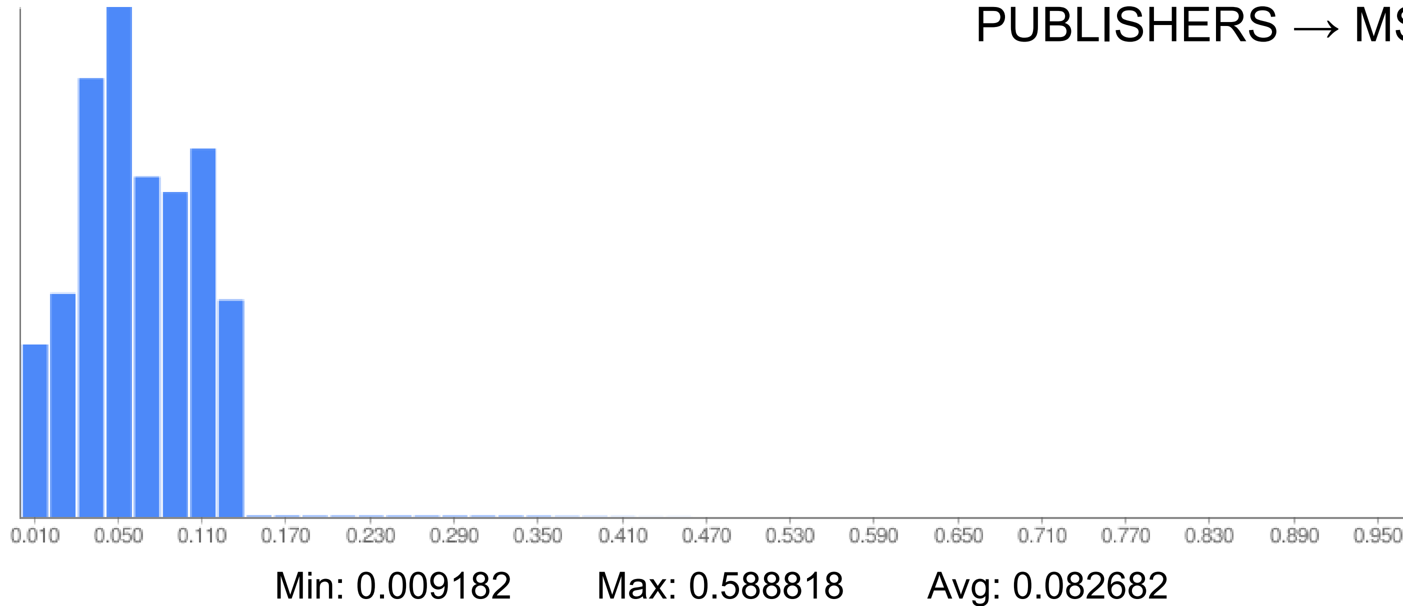


TEST SCHEMA AND PERFORMANCE METRICS

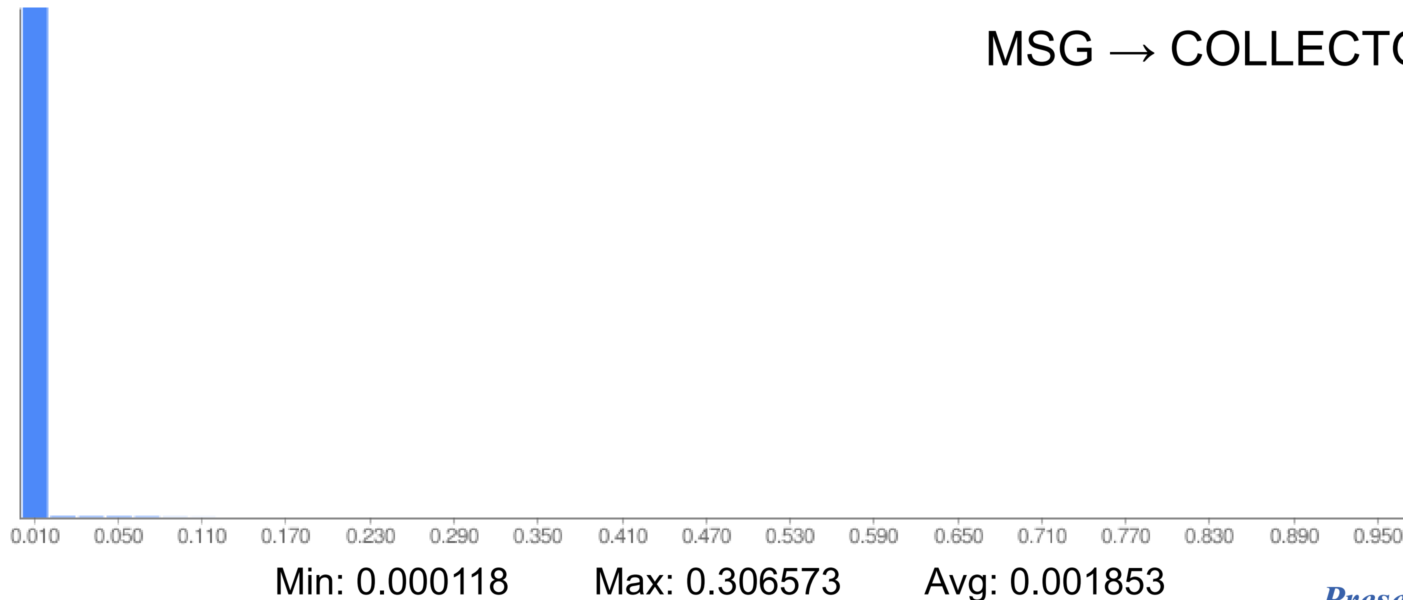
- Test scenarios:
  - Message length: 1KB, 16KB, 6KB, 128KB, 256KB
  - Frequency: 0.25Hz, 1Hz, 2Hz, 4Hz, 8Hz
  - Publishers: 50, 100, 400, 800, 1600
- Metrics:
  - Publisher to MSG server latency
  - MSG server to collector latency
  - # messages lost
  - # messages out of order

SYNTHETIC TESTING RESULTS

50 publishers, 8 messages (1KB) / s

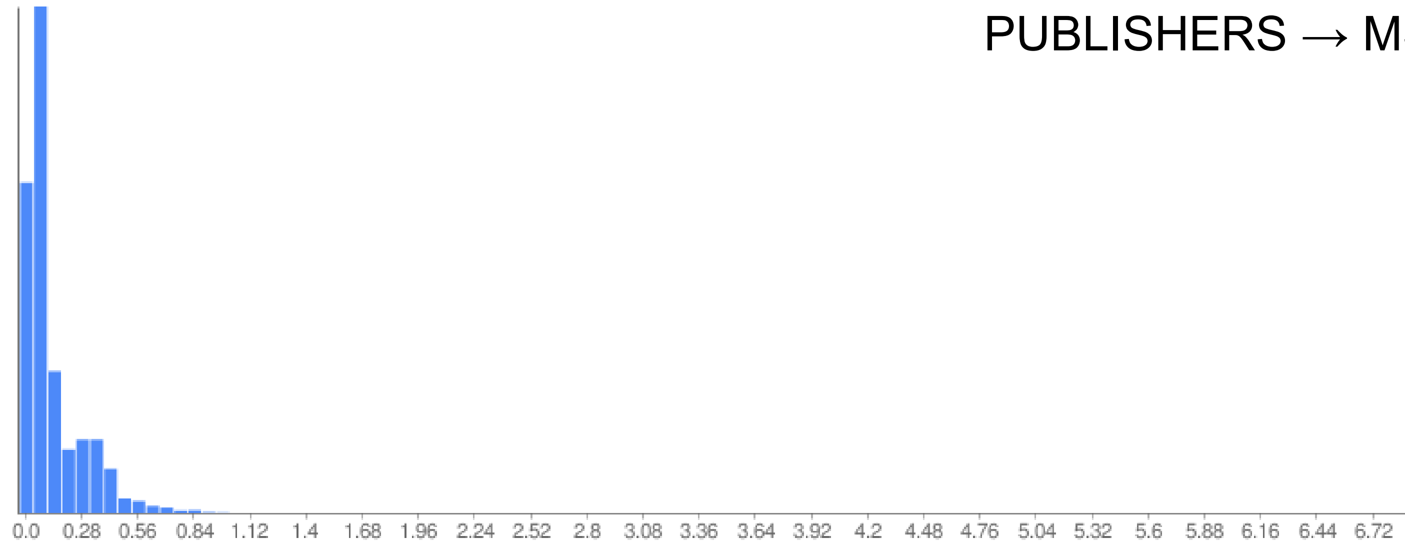


Min: 0.009182 Max: 0.588818 Avg: 0.082682

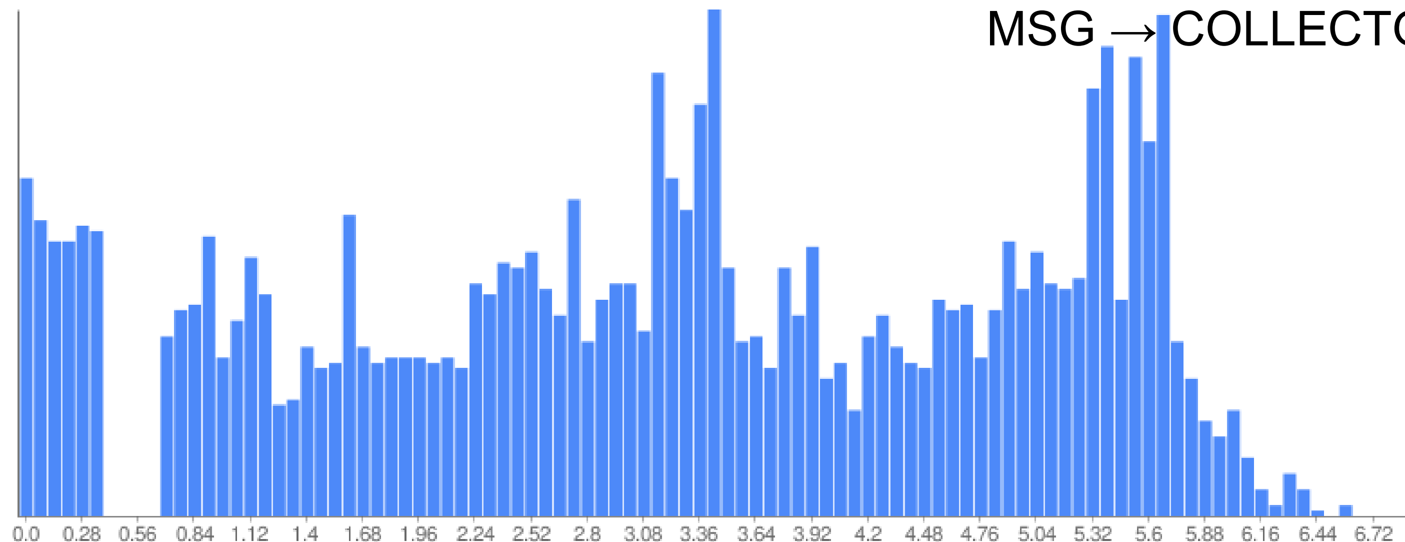


Min: 0.000118 Max: 0.306573 Avg: 0.001853

50 publishers, 8 messages (16KB) / s

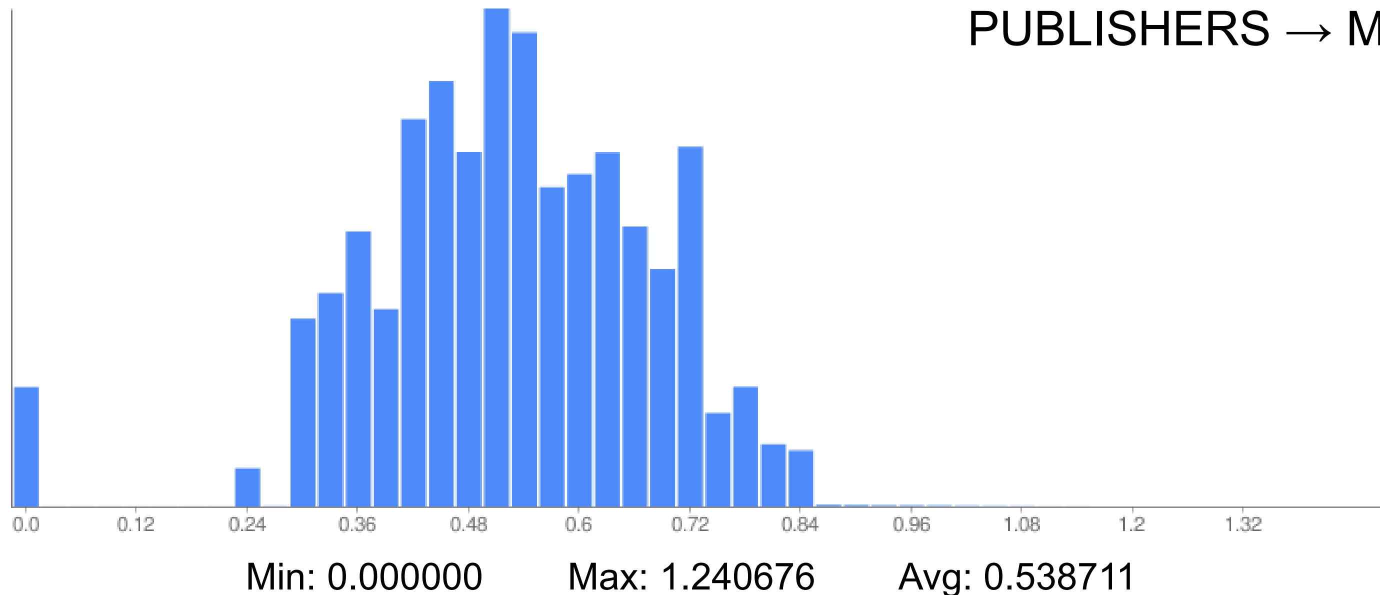


Min: 0.009197 Max: 0.987311 Avg: 0.159251

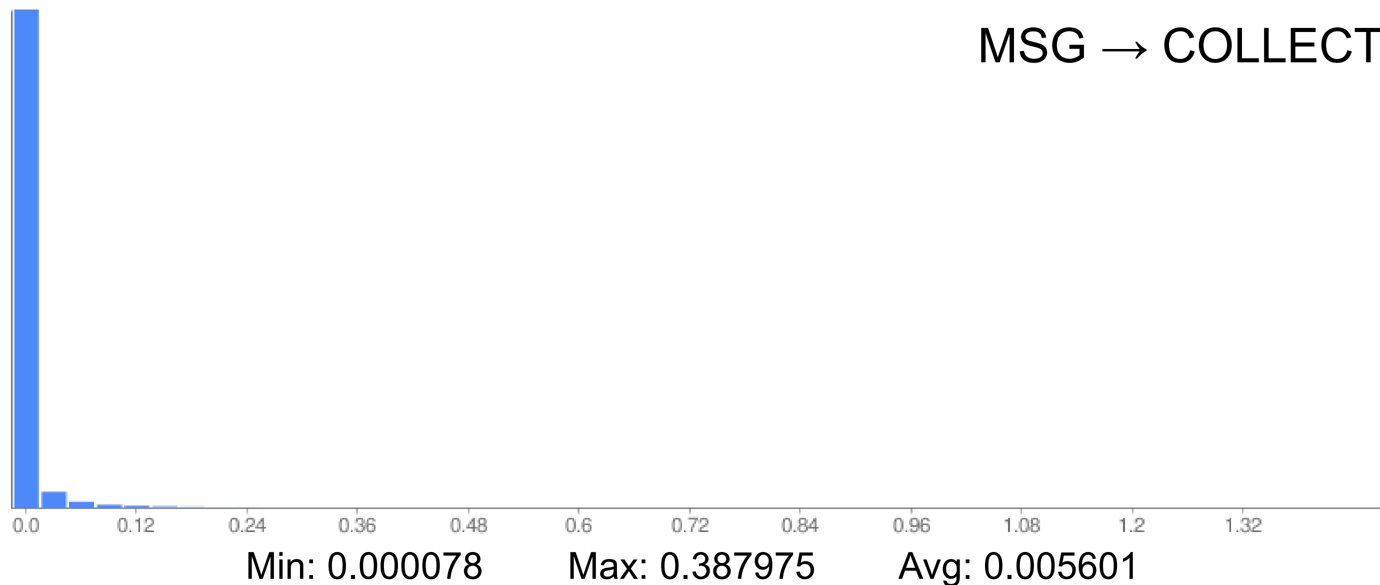


Min: 0.001815 Max: 6.637409 Avg: 3.237726

300 publishers, 8 messages (1KB) / s

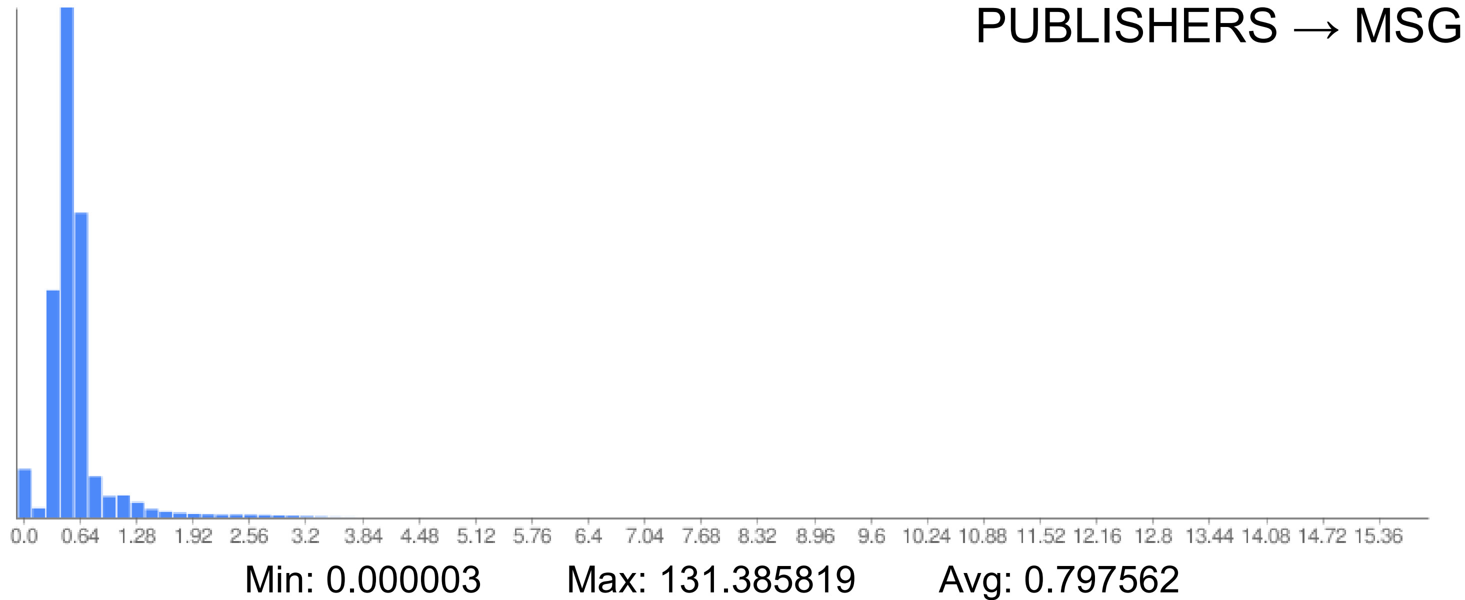


Min: 0.000000 Max: 1.240676 Avg: 0.538711

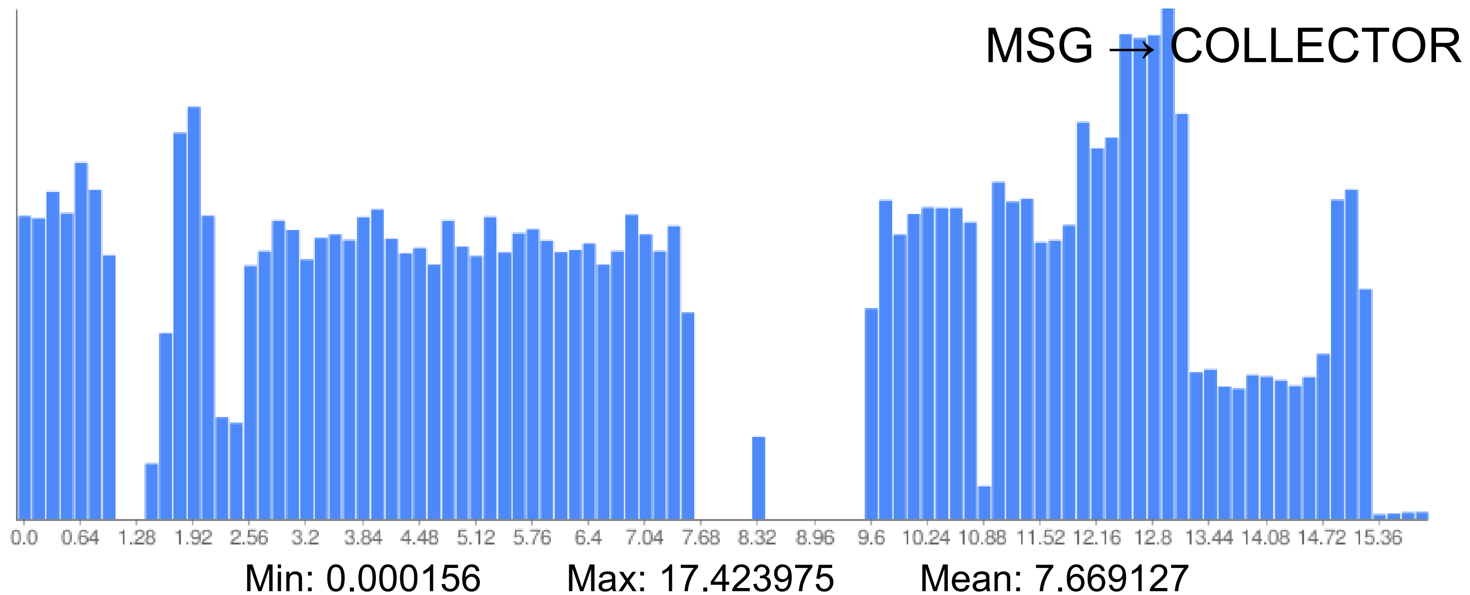


Min: 0.000078 Max: 0.387975 Avg: 0.005601

800 publishers, 4 messages (1KB) / s MSG crash



Min: 0.000003 Max: 131.385819 Avg: 0.797562

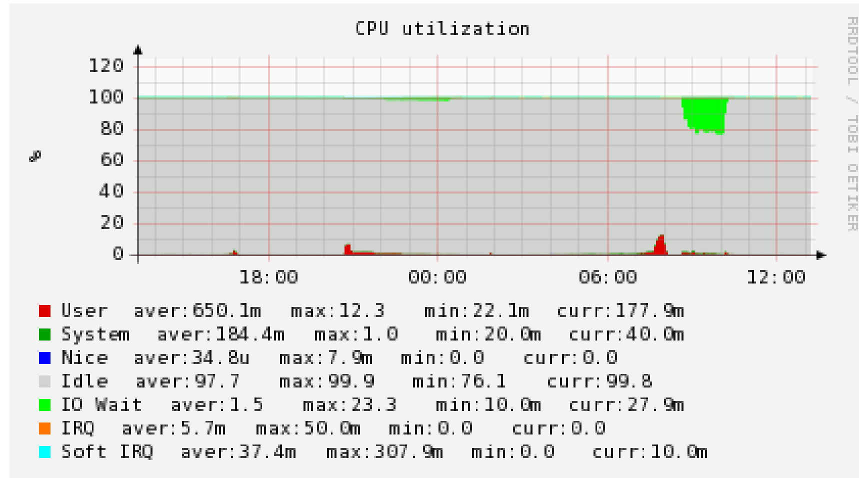


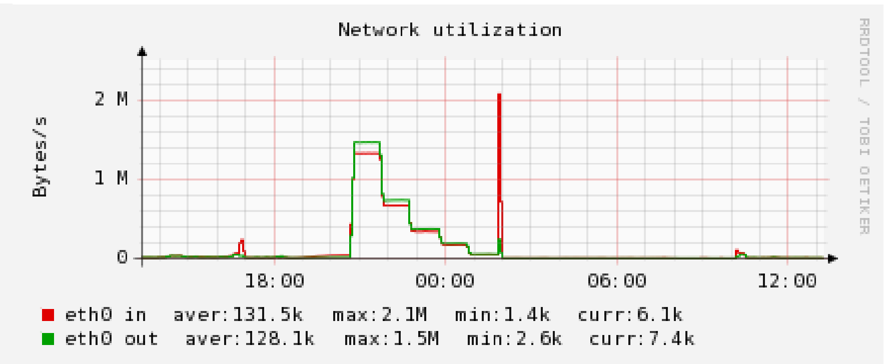
Min: 0.000156 Max: 17.423975 Mean: 7.669127

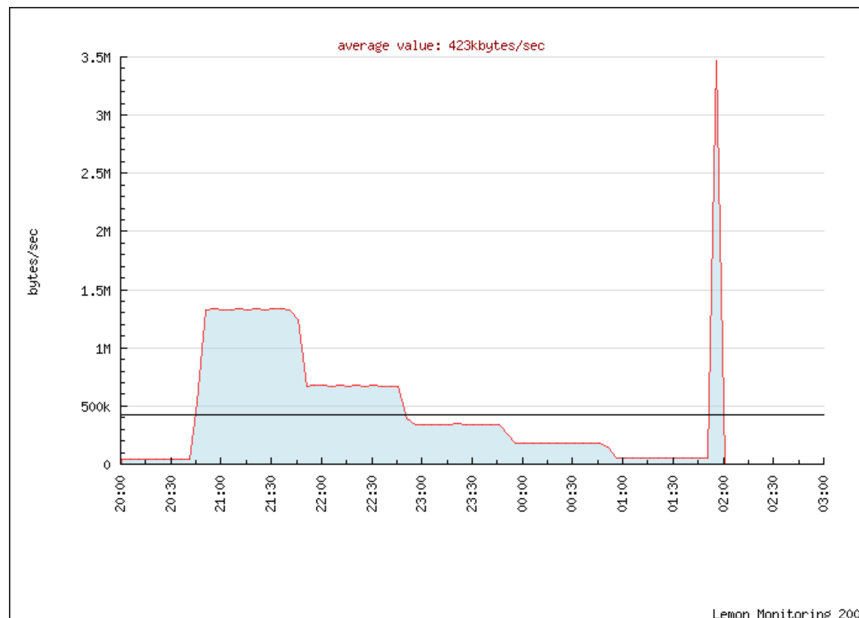
Grid Support

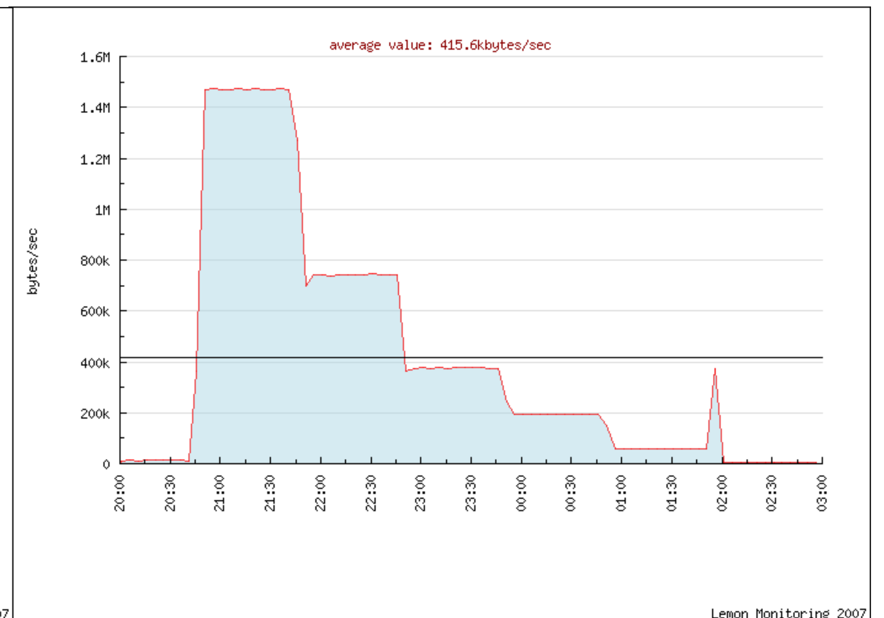
PERFORMANCE BOTTLENECK / MSG CRASH

CERN IT Department





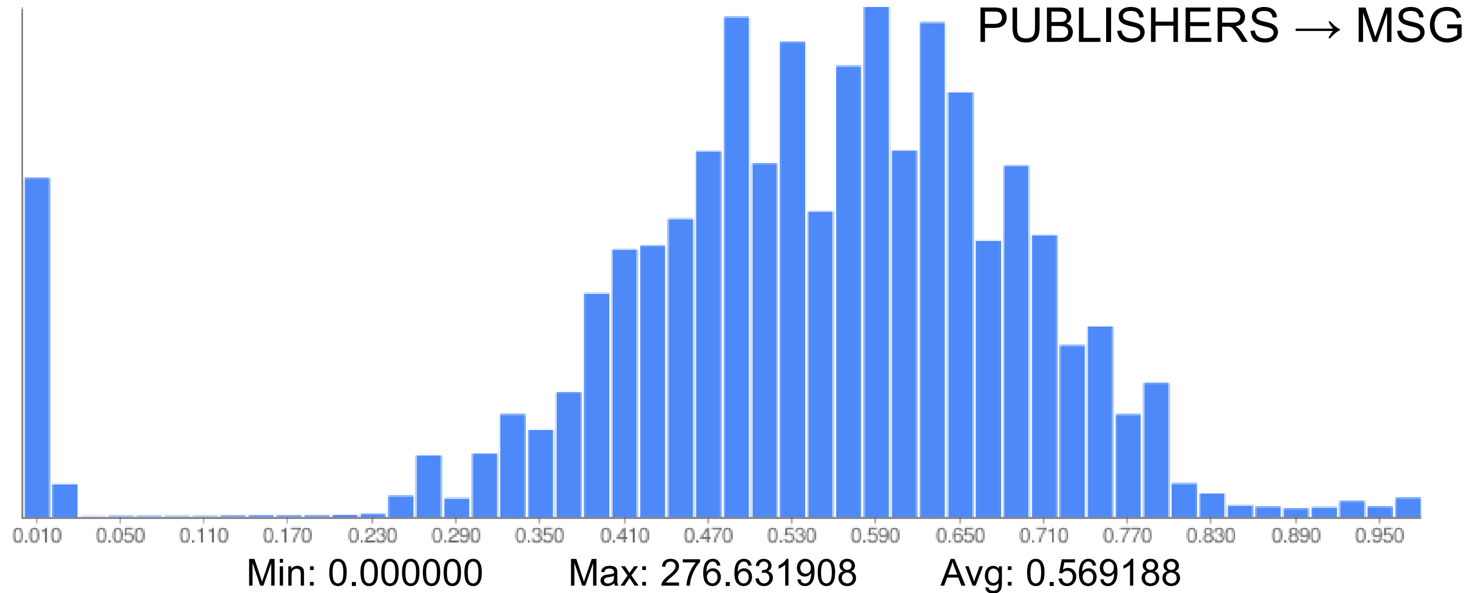




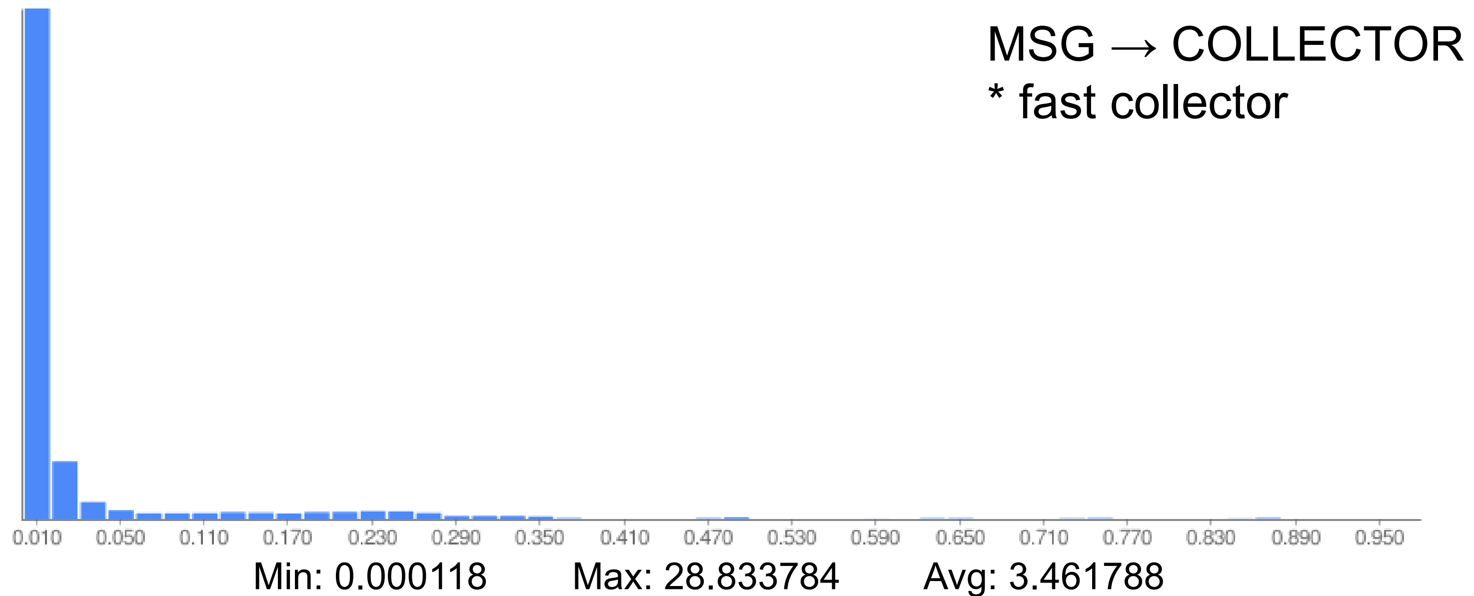
TWEAKING THE SERVER AND CONSUMER

- MSG SERVER:
  - Paging to memory
    - increase the available memory
    - example: 4GB instead of 1GB
  - Paging to hard drive
    - in addition to paging to memory
    - example: 10GB HDD temporary storage
    - disadvantages: doesn't necessairly solve the problem, may signifficantly decrease performance (~25%)
- CONSUMER
  - Higher throughput possible with advanced setup
  - Example: 3 consumers, instead of one, consuming messages on a round-robin basis (3x throughput as a result)

800 publishers, 4 messages (1KB) / second

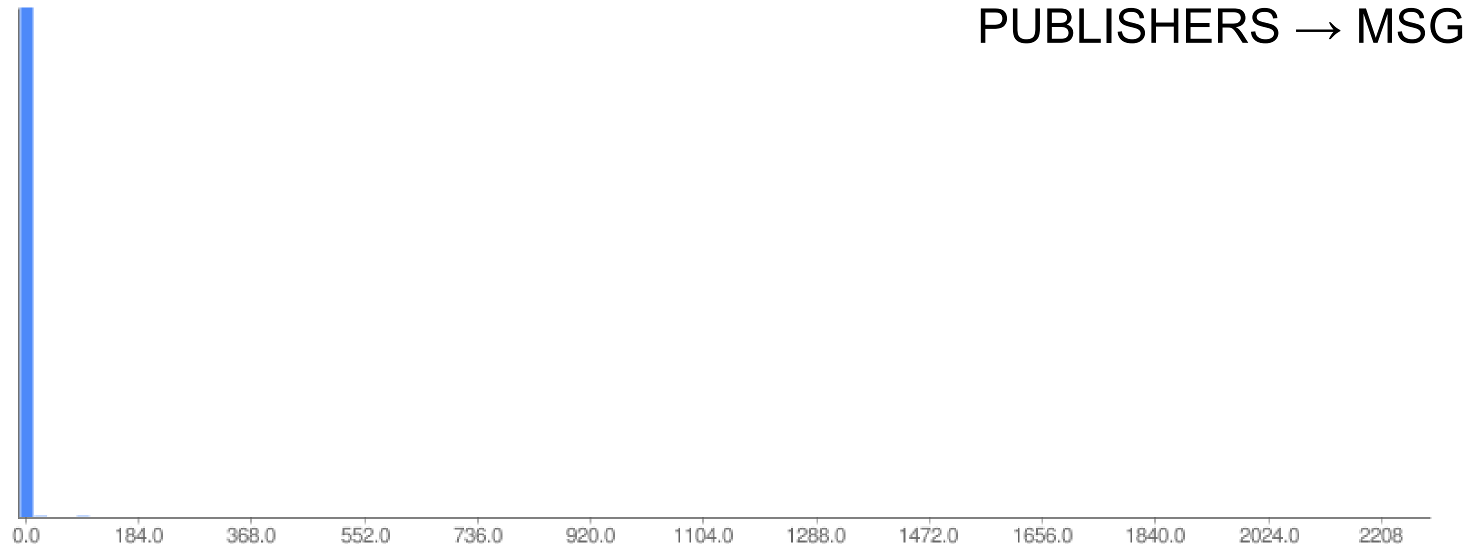


Min: 0.000000 Max: 276.631908 Avg: 0.569188

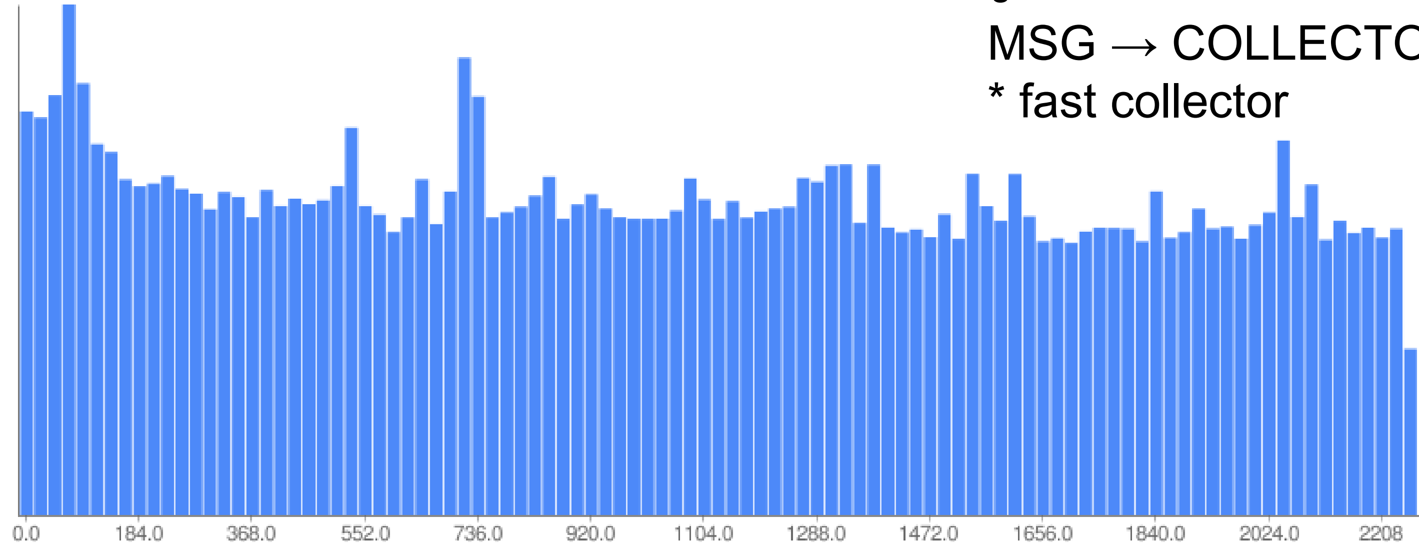


Min: 0.000118 Max: 28.833784 Avg: 3.461788

800 publishers, 8 messages (1KB) / s STOMP.py + Python limit



Min: 0.000000 Max: 124.415262 Avg: 1.030838



Min: 0.000000 Max: 2269.620000 Avg: 1087.622495

CONCLUSIONS

- Lightweight server (low CPU usage under heay traffic)
- Extensive testing with real world applications (HammerCloud STEP09, LQCD, Geant4)
- Extensive synthetic testing (more than 100M messages sent)
- No messages missing, no messages out of order
- Native Python client implementation scales well up to ~3500 1KB messages / second
- Higher performance possible with advanced setup (more consumers) or different acknowledge modes
- Choice between lightweight (Python / Ruby / Perl) and feature-rich, native (Java / C / C++) libraries