COSMOS/ORBIT Data Collection Tutorial

Orbit Measurement Library (OML)

What is it?

- An application instrumentation tool that allows collection of data from multiple nodes
- Client - Server implementation
- Minimal collection overhead
- Collect measurements to a single remote collection point
- Time-stamped push-based architectured

Some requirements

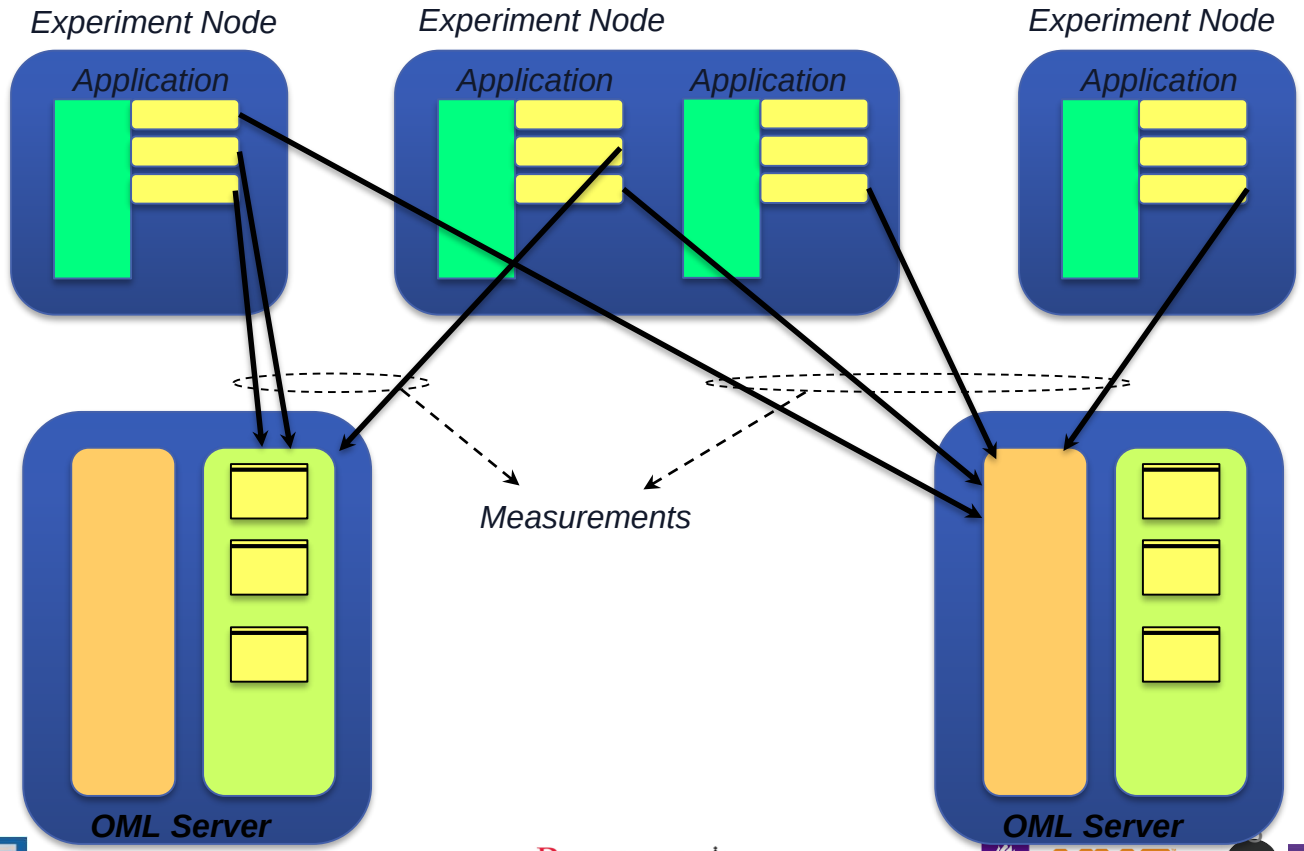
- dedicated NIC card for control and instrumentation to avoid experiment traffic interference
- c/c++ applications compile against liboml2

Use this site as reference for OML

– https://oml-doc.orbit-lab.org/

OML design goal: distributed software infrastructure to collect measurements in real-time while providing flexible (and dynamic) way to modify the collection process.

OML – Measurement Collection



OML2 C++ API Outline

Steps involved to use OML Wrapped API in a C++ application.

1. Create OML object
   CWriteOML() oml;

1. Start by initializing the object with required parameters
   oml.init(std::string oml_id, std::string oml_domain, std::string oml_collect);

oml-id	this is the sender to the measurements
oml-domain	database filename where recorded values will be stored
oml-collect	network location of oml server

OML2 C++ API Outline (cont)

3. Register measurement points - specify a variable name it's data type.

   oml.register_mp("RadioName", OML_STRING_VALUE);
   oml.register_mp("PowerInDb", OML_DOUBLE_VALUE);

3. Start the collection daemon - this opens a database file and creates a schema including the registered measurement points.

   oml.start( );

OML2 C++ API Outline (cont)

5. Update measurement point value with associated value

```
// a container of measurement points
double d64_power = radio.value();
oml.set_key("RadioName", (void*)string.c_str() );
oml.set_key("PowerInDb", (void*)&d64_power );
```

5. Send measurement points to database.

```
oml.insert();
```

Data collection & recovery

Outline an existing application that is integrate with OML.

- Use C++ wrapped OML API.

Run OML enable application & send measurements to server.

Recover data from database file.

- Access database directly.
- Parse database file to view results.

Run application from inside node.

COSMOS Summary

- Focus on ultra high bandwidth, low latency, edge cloud
- Open platform (building on ORBIT) integrating mmWave, SDR, and optical x-haul
- 1 sq mile densely populated area in West Harlem
- Local community outreach
- Research community:
  - Develop future experiments, provide input
  - (short term) get involved in the educational outreach

More information:

http://advancewireless.org http://www.orbit-lab.org http://www.cosmos-lab.org
http://omf.orbit-lab.org http://oml-doc.orbit-lab.org

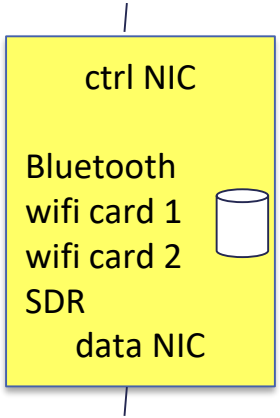
COSMOS RUTGERS COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK NYU NYC <sh> The City College of New York ARIZONA IBM

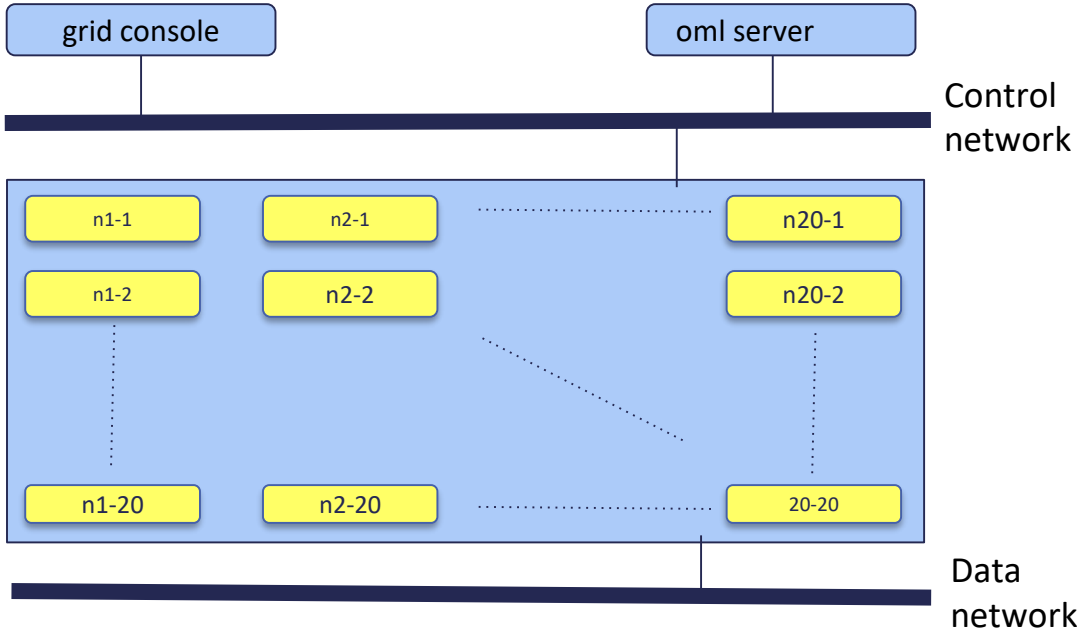
Appendix

Supplementary information

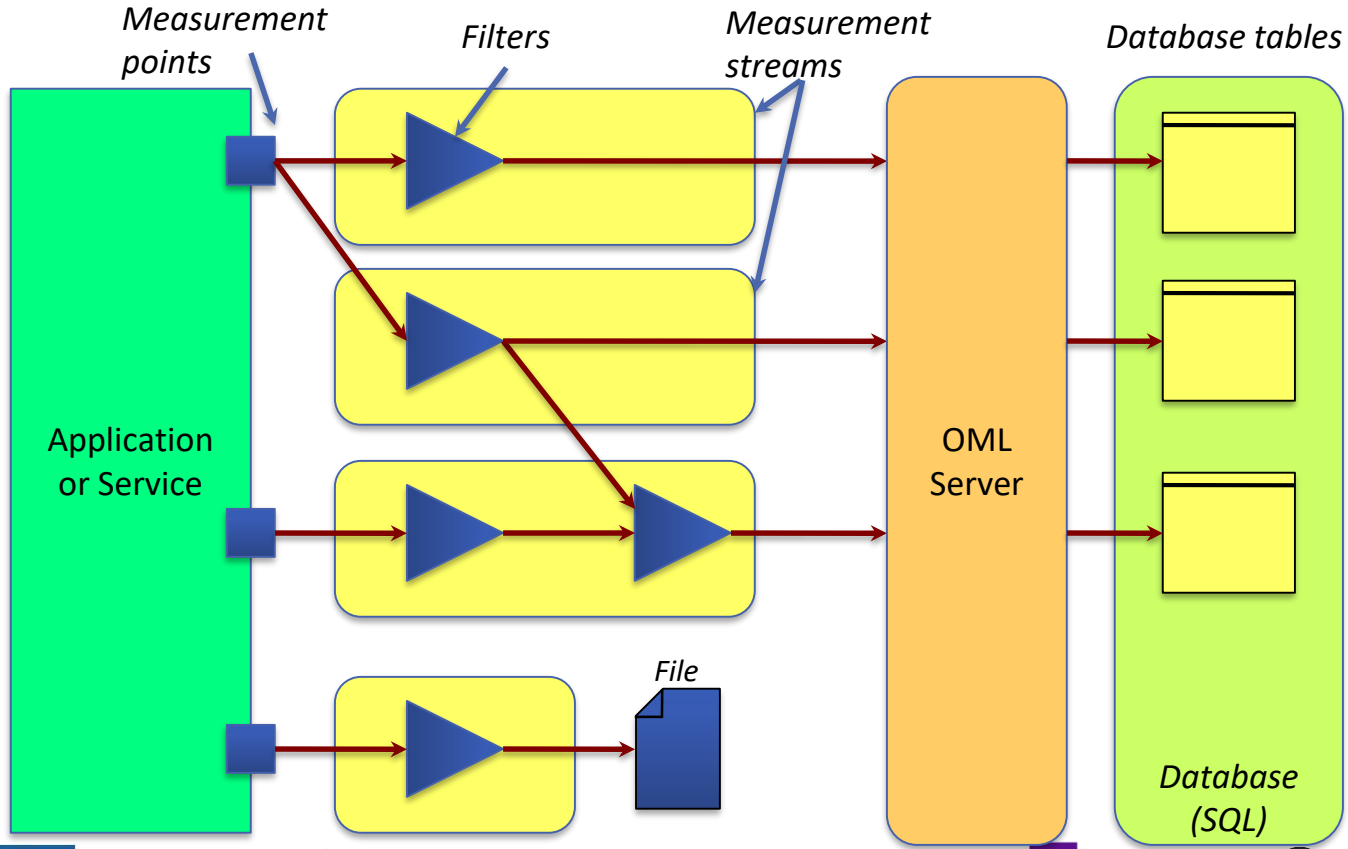
- OML description and advanced functionality

Simplified diagram





OML Client + Server



Experimental Support

Applications

- Traffic Generation/Measurements
  - OTG … Traffic Generator
  - Iperf
- Monitoring
  - Libtrace
  - Libsigar
  - Spectrum Analyzer
  - GPS
  - (Weather)
- Components
  - TinyOS/Motes
  - (GnuRadio)

Filters

- Plug-in Architecture
- User extensibility

Current List

✓ Stddev
✓ Average
✓ First
✓ Histogram

OML2 Base Functions

- **omlc_init()** – used for OML initialization
  omlc_init(arg(0), &argc, argv, o_log);
- **oml_add_mp()** – called to register each measurement point with the container
  oml_mp = omlc_add_mp("udp_out", oml_def);
- **omlc_start()** - used to start the local collection daemon
- **omlc_inject()** – used to send all measurement points to the oml server for storage
  omlc_inject(oml_mp, values_container);