Real Time Data Event Streaming & Processing (RESP)

(Presentation to DCAE Team)

Habib Madani  Habib.Madani@Huawei.com
Parviz Yegani  Parviz.Yegani@huawei.com

June 7, 2018
RESP Introduction
Why RESP?

**ONAP Current Operation**

- Consumers
- Message Server
- Collect and push to Message Bus for Pull
- API-based end-point collection
- End-Point

**RESP Pipeline**

- Real-time Data Event Stream
  - Collection
  - Aggregation, Filter
  - Transformation
  - Consumers
  - End-Point

(Use Case II) Real-time Pub/Sub and Processing

- Pub/sub Event
  - DB/cache agent

- Modeled logical data objects based pub/sub stream achieve declarative system through auto synchronization

(Use Case I) Real-time Asynchronous Pub/Sub

- Dynamic strongly typed event message-based pub/sub

Data information embedded in message as text

Static API/REST Pull

ONAP Current Operation

RESP Pipeline
RESP Proposal

Building on DCAE Base Platform (DCAEGEN3)

DCAEGEN3: Focus on building a platform which is adaptable to changing environment

- Data Collection: interfaces keep evolving – Data Streaming, gRPC, etc
- Analytics: AI, Machine Learning, Capacity, Cost, Perf., … (real value-add)
- Common Platform: Network sharding, Wireline, IOT, Cloud, etc
- ONAP Platform is used as a base to build from
- Flink: Real-time event streaming Analytics
- DDS: Real time Data centric middleware
- Beam: An advanced unified (Batch + Real-time) programming model
- Design-Time / Run-Time separation principles
- Microservices-based architecture
- Data Collector: collects data from the element (network elements, IOT apps,..)
  - Events/traps, statistics, logs, control plane, data plane
- Data Storage: collected data will be stored
- Pipeline Design: transforms raw data into meaningful data
RESP APIs

- RESP a new functional entity for DCAEGEN3 Platform
- RESP can co-exist with existing Bus and ONAP components
- RESP impacts ONAP Design-time and Run-time using ONAP APIs (with extensions as necessary)
- All ONAP components potentially can communicate in real-time with common library

NOTE:
- IoT/5G device/VNF can push DDS-based event stream
- Proxy can push DDS-based event streams with legacy backend interface
Data Analytics and Processing Pipeline

Natively with RESP, mechanisms are provided to:

0. Native ONAP APIs leveraged with RESP related metadata and model extensions/additions

1. Ingress, **Exactly-Once and Eventually Consistent** synchronization mechanism available with DDS-based event streaming between peers.
   - DDS provides fine-grained control over the QoS setting for each of the entities involved in the system.
   - Ingress and Egress Integration with DMaaP/Kafka Bus and other ingestion needs
     - Non-blocking and concurrent routing/handling of data streaming (Akka)
     - Handle bursty stream processing, in parallel and quickly (Beam -> Flink)

2. Egress real-time DDS-based data stream publish
Data Collection Layer
DDS vs Socket API Comparison
# DDS and Sockets

<table>
<thead>
<tr>
<th>DDS</th>
<th>Sockets</th>
<th>DDS Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbuilt Data Domain capability for scoping of Data across the collection space</td>
<td>N/A</td>
<td>Allows for data space segregation across the collection point/network built on “Domain” concept. Allow partitioned collection points through these “Domains”</td>
</tr>
<tr>
<td>Asynchronous, primarily not connection Oriented follows Pub/Sub</td>
<td>Connection Oriented &amp; Synchronous</td>
<td>DDS is optimized for UDP with its own wire protocol. It can leverage TCP as well. Further if multicast is available then it excels at using multicast natively</td>
</tr>
<tr>
<td>1-1, 1-n and n-m supported Pub/Sub</td>
<td>Point-to-Point</td>
<td>Flexible deployment model, allows for deployment across sharded Domains</td>
</tr>
<tr>
<td>DDS can perform fast data movement with bounded latency, linearly scale for both publishers and consumers in a distributed global space with reliability For happy path performance does degrade if any failure occurs</td>
<td>No reliability data communication beyond the TCP protocol Failure management for socket API is quite challenging</td>
<td>DDS allows tracking of fast and slow publishers/consumers and handle the data events with “Eventual Consistent” delivery (re-try mechanism is handled transparently by DDS)</td>
</tr>
<tr>
<td>Dynamic/Auto Discovery, DDS handles discovery and management of endpoints</td>
<td></td>
<td>Latency and Performance is unmatched. As such our lab tested experience has shown very high throughput for DDS based messaging. It also depends on the message size and the network pipe to which DDS performed well. The following link from Adlink and RTI also highlights the performance numbers: <a href="http://www.prismtech.com/vortex/vortex-opensplice/performance">http://www.prismtech.com/vortex/vortex-opensplice/performance</a> <a href="https://www.rti.com/products/dds/benchmarks#CPPLATENCY">https://www.rti.com/products/dds/benchmarks#CPPLATENCY</a></td>
</tr>
</tbody>
</table>

**Dynamic/Auto Discovery, DDS handles discovery and management of endpoints**

Simple code:
```
DomainParticipant dp(1);
Topic<NFV_VES> topic("VES_PM");
Publisher pub( dp);
DataWriter<NFV_VES> dw ( pub, topic);
```

**Client-Server (Clients and Servers need to be aware of each other)**

This allows for very efficient deployment model, shields from topology changes and provides a plug-play approach
## DDS and Socket

<table>
<thead>
<tr>
<th>DDS</th>
<th>Sockets</th>
<th>DDS Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDS has inbuilt capabilities for communication behaviors like filtering, durability, saving historical data, detecting deadline misses and changes in liveness, and many more through QoS</td>
<td>N/A</td>
<td>All these are configurable to be turn on/off depending on the nature of the endpoints. These features allow you to collect data with fine-grain precision, and also allow data traceability for real-time management. Application QoS configurations allow for prioritizing stream tracking.</td>
</tr>
<tr>
<td>For real-time, you need something that can differentiate between reliable and best-effort, which means using light UDP or similar unreliable transport, with an intrinsic reliability protocol offered on top of the unreliable transport</td>
<td>Full Lifecycle of the connection needs to be instantiated and maintained in the clients and servers</td>
<td>For real-time, you need something that can differentiate between reliable and best-effort, which means using UDP or similar unreliable transport, with an intrinsic reliability protocol offered on top of the unreliable transport.</td>
</tr>
<tr>
<td>Polyglot bindings</td>
<td>N/A</td>
<td>Allows for multiple language bindings for Event stream Pub/Sub.</td>
</tr>
</tbody>
</table>

### Diagram

- **VNF**
  - ... 103 102 101
- **Socket >> 10 msec**
  - 100 99 98 97
- **VES Collector for < 5 msec**
  - 96 95 94 ...

**DDS Time t (5 msec)**

**sec QoS**
Domain-Driven Distributed Deployment Use Case

Domain 0

Edge DCAE

Central DCAE

Domain 1

Domain 2

DDC PUB/SUB

VES Collector

N nailed socket VES connections

N Loosely coupled pub/sub VES topic event streams
DDS Publisher Side Code

```cpp
#include <dds.hpp>

int main(int argc, char** argv) {
  DomainParticipant dp0;
  Topic<Meter> topic("SmartMeter");
  Publisher pub(dp0);
  DataWriter<Meter> dw(pub, topic);

  while (!done) {
    auto value = readMeter();
    dw.write(value);
    std::this_thread::sleep_for(SAMPLING_PERIOD);
  }

  return 0;
}
```

Socket Server Side Code

```c
// Server side C/C++ program to demonstrate Socket programming
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#define PORT 8000
int main(int argc, char const *argv[]) {
  int server_fd, new_socket, valread;
  struct sockaddr_in address;
  int opt = 1;
  int addrlen = sizeof(address);
  char *hello = "Hello from server";

  // Creating socket file descriptor
  if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
    perror("socket failed");
    exit(EXIT_FAILURE);
  }

  // Forcefully attaching socket
  if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
  }

  address.sin_family = AF_INET;
  address.sin_addr.s_addr = INADDR_ANY;
  address.sin_port = htons(PORT);

  // Forcefully attaching socket
  if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0) {
    perror("bind failed");
    exit(EXIT_FAILURE);
  }

  if (listen(server_fd, 3) < 0) {
    perror("listen");
    exit(EXIT_FAILURE);
  }

  if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))<0) {
    perror("accept");
    exit(EXIT_FAILURE);
  }

  valread = read(new_socket, buffer, 1024);
  printf("%s\n",buffer);
  send(new_socket, hello, strlen(hello), 0);
  printf("Hello message sent\n");
  return 0;
}
```
RESP for Casablanca (Normal Path)

0. DCAE Control components
   • Driven through new TOSCA blueprints by Cloudify/Orcestration
1. DDS Enabled with VES Event Streaming Topics
2. VES Collector
3. RESP Service Components
   • Use Case Microservices
4. RESP Platform Components
   • Flink with Beam API
5. RESP Microservices interact with DMaaP/Kafka

Beyond Casablanca (Fast Path)

1. DDS Enabled with VES Event Streaming Topics
2. VES Collector
3. RESP Service Components
   a) Use Case Microservices Actors (root, workers)
   b) Routing to Error or Happy Paths
4. RESP Platform Components
   a) Base I/O Processing
   b) Beam API calls and Processing
5. RESP Microservices interact with DMaaP/Kafka
6. RESP Microservices interact with DDS
   a) RESP Msrv. worker actor egress DDS stream publication
   b) Potential Southbound ingress stream
RESP Functional Description for Casablanca

Definition:
RESP is ONAP/DCAE subsystem that enables Fast Data Event Streaming, Fast Data Event Stream based higher level analytics and correlation for business and operations activities. RESP functionality will allow for fast data based collection of performance, usage and configuration data; provides handling of fire-hose event source in real time along with analytical processing for supporting operational decisions, trouble shooting and management; provides fast track results through pub/sub data event stream publications to rest of the ONAP system for FCAPs and other functionality.

Provided Interfaces:
- Interface 1: Data collection interface (provided by DCAE collectors, consumed by VNFs and others)
  - Interface for various FCAPs data entering DCAE/ONAP.
  - Interface for triggering the deployment and changes of a control loop
- Interface 2: Deployment interface (provided by DCAE Deployment Handler, used by CLAMP and other northbound applications/services)
  - Interface for querying the deployment and information of the services that are registered to DCAE Consul
- Interface 3: Configuration Binding Service
  - Interface for querying the information of the services that are registered to DCAE/RESP Consul
- Interface 4: Data collection interface (provided by RESP/DDS collectors VES, Proxy and others)
  - Interface for various FCAPs, IoT other data entering DCAE/ONAP.
  - Interface for triggering the deployment and changes of a control loops, IoT TCA and other use cases
- Interface 5: Deployment interface (provided by DCAE/RESP Deployment Handler, used by IoT and other northbound applications/services)
  - Interface for querying the information of the Fast Event based services that are registered to DCAE/RESP Consul
- Interface 6: Configuration Binding Service
  - Interface for querying the information of the services that are registered to DCAE/RESP Consul

Consumed Interfaces:
- Interface 1: Data movement platform interface (provided by DMaap)
  - Interface for data transportation between DCAE subcomponents and between DCAE and other ONAP components
  - This interface can also be used for publishing events to other ONAP components.
- Interface 2: Data enrichment interface (provided by A&AI)
  - Interface used by DCAE collectors and analytics for querying A&AI for VNF information for the purpose of enriching collected raw data by adding information not contained in original data.
- Interface 3: Service model change interface (Provided by SDC)
  - Interface for DCAE/RESP Service Change Hande fetching control loop models and model updates for RESP apparatus (Akka enabled microservices, BEAM SDK, Flink, TSDB)

Consumed Models: TOSCA models describing IoT & control loop construction (e.g. collection and analytics apparatus)
What is RESP?

Real-time data event streaming and processing pipeline comprised of:

• Real-time Event Streaming Middleware
  - QoS-based reliable real-time end-to-end event streaming
  - Non-blocking, highly available collection and routing of events
  - Asynchronous pub/sub based communication
  - Consistent synchronization to accurately represent network states
  - Model-driven approach through data models for consistent structure

• Real-time Processing for Data Analytics
  - Efficient and reliable processing of data in motion
    (via reducing a huge amount of data influx through tiered deployment)
  - In-memory very fast, efficient and reliable processing

• Geo-data Store and Synchronization
  - Services that need states preserved with consistent geo-data event store and event streaming-based data synchronization
<table>
<thead>
<tr>
<th>DDS Components</th>
<th>DDS for Realtime</th>
<th>Data Distribution BUS</th>
<th>Akka+Kafka for HA Collection</th>
<th>Beam &amp; Flink for Transformation &amp; DB Persistence Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESP Components</td>
<td>Cloud Native</td>
<td>Framework</td>
<td>Analytics Framework</td>
<td>DDS Cache/Event Store</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Real-time Streaming and Batch collectors</td>
<td>Unified Data Streaming Programming Model andDSL and pipelining</td>
<td></td>
</tr>
<tr>
<td>P2P Tiered Geo Distributed PUB/SUB (Blood Stream)</td>
<td>Rich QoS support for reliability, extreme low latency support through light wire protocol</td>
<td>Concurrent, Actor model, Non-blocking</td>
<td>Realtime Streaming and Batch Processing</td>
<td></td>
</tr>
<tr>
<td>Asynchronous/Synchronous</td>
<td>Distributed, Resilient, parallel processing, Back Pressure</td>
<td>Collection/Aggregation possible from many streams</td>
<td>Rich Parallel processing with rich API</td>
<td></td>
</tr>
<tr>
<td>Strong Typing(binary), data model/IDL</td>
<td>Load Balancing and Partitioning with reliability, HA and error handling</td>
<td>Time Series and Other</td>
<td>Resilient Distributed Data processing, with in-memory Cache analytics for huge stream input</td>
<td></td>
</tr>
<tr>
<td>Inherent Security</td>
<td>Discovery, Query and Filter</td>
<td>Connectionless optimized</td>
<td>ML analytics &amp;Graph/R predictive modeling and analytics</td>
<td></td>
</tr>
<tr>
<td>Streams/Events/Logs/Payload</td>
<td>Connectionless optimized</td>
<td>Publisher/Consumer through a Relationship model</td>
<td>Rich Query and Time Series</td>
<td></td>
</tr>
<tr>
<td>Asynchronous/Synchronous</td>
<td>Support for reliability, extreme low latency support through light wire protocol</td>
<td>Reduce Chatter</td>
<td>Beam allows integration with other runners (Spark, Storm) for compatibility</td>
<td></td>
</tr>
<tr>
<td>DDS Cache/Event Store</td>
<td>Need based DB use for backend store</td>
<td>Geo Distributed</td>
<td>Backend Connectors for integration of huge K,V store</td>
<td></td>
</tr>
<tr>
<td>DDS Cache/Event Store</td>
<td>Geo Distributed</td>
<td>Consistency( ✓ Eventual) for supporting statefull entities</td>
<td>Backend Connectors for integration of huge K,V store</td>
<td></td>
</tr>
</tbody>
</table>

- DDS for Realtime
- Data Distribution BUS
- Akka+Kafka for HA Collection
- Beam & Flink for Transformation & DB Persistence Storage

**Real-time Streaming and Batch collectors**
- Real-time Streaming and Batch collectors
- Concurrent, Actor model, Non-blocking
- Distributed, Resilient, parallel processing, Back Pressure
- Collection/Aggregation possible from many streams
- Load Balancing and Partitioning with reliability, HA and error handling
- Time Series and Other

**Unified Data Streaming Programming Model andDSL and pipelining**
- Realtime Streaming and Batch Processing
- Rich Parallel processing with rich API
- Resilient Distributed Data processing, with in-memory Cache analytics for huge stream input
- ML analytics &Graph/R predictive modeling and analytics
- Rich Query and Time Series
- Beam allows integration with other runners (Spark, Storm) for compatibility
Geo Distributed Global Data Centricity and Synchronization
Low latency Pub/Sub P2P, Broker(optional) and N2M
Push to Subscribers
Query and Filter on data-value
Data Centric
Best effort, last-n, reliable and exactly-once
20+ QoS settings

Broker Based
Pull from Cluster
Opaque Data
Message Centric
At least once, at most once, exactly once
Controlled reliability and durability

Comparison: DDS vs Kafka
RESP Interfaces and API calls

Configuration & Operations
- Artifact design and creation mapping to model-driven network services/VNFs for pub/sub topics (e.g., VES and other ingestion sources)
- A&AI – resource inventory
- Service and resource orchestration
- Ongoing real-time updates for new services and/or changes to existing services

Run-time Operations
- Real-time event streams based on pub/sub configurations from VNF
- Main application Akka actor-based concurrent handling of streaming event with DDS embedded library real-time event streams, and cache DB store
- HA and error handling
- Real-time Beam SDK calls to Flink real-time analytics tools

Interfaces
- API calls to push configuration information to the DCAE controller
  - Pub/Sub topic push
    - Analytics
    - Filter, Search, Group by resources
    - Rules for actions for component interactions
- Input run-time real-time event streams
- Output real-time event streams
- API calls (if needed)
ONAP Gaps

Real-time Data Streaming and Data Processing Pipeline

1. Real-time data model-based event streaming for north to south
2. Collection real-time data event stream and routing of stream for actionable processing
3. SQL operations capability on the real-time data stream
4. Exactly once streaming of real-time event data for sequential processing
5. Fire hose event data streams processing in real-time and reliably
6. ONAP applications enablement through real-time library-based pub/sub

Rich Complementing Common Service Beyond a BUS
## ONAP Gaps

<table>
<thead>
<tr>
<th>MSB BUS</th>
<th>DMaaS /Kafka</th>
<th>DDS Real Time Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microservices Operations API centric</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>JMS capability to handle microservices health operations</td>
<td>Pub/Sub <strong>Message</strong> oriented middleware, JSON based payload, centralized</td>
<td>Provides a rich Model driven binary <strong>Data centric</strong> Pub/Sub Geo <strong>Distributed</strong> Domain based capability to provide full microservices interconnectivity beyond Java</td>
</tr>
<tr>
<td>No Native Data Synchronization</td>
<td>No Native Data synchronization</td>
<td><strong>Logical data Objects</strong>, changes automatically track to make the system track states and <strong>self synchronizing</strong></td>
</tr>
<tr>
<td>Client/server</td>
<td>TCP</td>
<td>UDP/TCP with a <strong>QoS</strong> based wire protocol for optimization and reliability fine grain flow control, <strong>for very fast binary data type delivery and tracking of events</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Concurrent, non-blocking, back-pressure handling end to end real-time pipelined system</td>
</tr>
</tbody>
</table>
**DDS Domain-based Network Sharding Example**

**Enterprise Use Case**

- **Achieve a declarative model**
- Virtual devices scale horizontally in vertical customers domains addressing demarcated handling for deployments, configuration, running and operations
- Isolate Issues by Honey potting
- Deploy using red-blue army approach
- Unify heterogeneous systems with strong typing support as ‘services’
- Multi-tenancy enabler
DDS for Geo Distributed Data Synchronization (DB)

- Figure 1 depicts cross DC deployment
  - Scenarios 1-4 reflect the network link state
  - The data flow across network tracked reliably for both Deterministic and Non-Deterministic network links
  - Geo Data synchronization guaranteed through Eventual Consistency for apps as shown in Figure to provide Geo distributed Event store
  - Light wire protocol allows for low latency times in the order of < 1msec over connectionless interconnect and with binary strongly typed data with QoS for reliability

- Figure 2 depicts a DDS based application deployment
  - Shows efficient query & filtering capability
  - Event DB store as Cache/persistent for logical objects
  - Very Rich meta-data information on each event
  - Each App has an embedded library
Unifying Fabric – Microservices Centralized and Edge Interconnect
(Logical Object based State Synchronization)

- MSB BUS Internal and External Event based BUS
- DDS fully supports intelligent Interconnect needs for IoT and SDN, PNFs, VNFs, a unifying fabric
- Logical local objects at the Publishers/Subscribers dynamically reflect/track Endpoint Telemetry/State/Other in real-time asynchronously
- Centralized NB systems, Controller/s, NE/VNFs, Service Discovery, E2E Operations all connected through DDS Data Domains
- A Key DDS based interconnects can be sharded and tiered per Pub/Sub to provide a microservices loosely coupled env.
VES Mapping

VES – Common Event Data Model

- Common Event Data Model
  - Common Header and Domain Specific Event
  - Extensible for additional fields or domains
  - Collector connection and data profile established at VM creation
  - Connection/authentication/profile parameters injected into VM
  - Data profile is fully controllable, to optimize telemetry overhead

VES Requirements Mapping

- Overall common Event Data model, Event Stream and collection architecture
- OPNFV support for the VES Common Event Data Model
  - Consisting of a Header and an Event Specific Block, with additional Name/Value fields for extensibility
- A VNF Event Stream (VES) Common Event Data Model
  - A VES Agent that can collect the VNF Event Stream data from the VNF and deliver it to the VES Collector
  - A VES Collector that can consume the VNF Event Stream data, and store the data in a database backend
- VES plugins for integration with OpenStack infrastructure services such as Monasca and Vitrage
- Common Event Data Model (Fault, Custom, programmable), authentication, access

DDS for Common Data Bus

- Common Data model with declarative IDL, Extensible types P2P Pub/Sub
- Same API exposed for all HW, OS
- Dynamic Discovery across Domains & Topics accommodating common header with profile creation through IDL, full control
- Rich QoS policies, allow for fine grain communication control beyond Transport (very low latency support, jitter support) - Deep dive for fine tuning
- Strong Type support, application read(), write() calls with specific data types, can support different payload with defined types (different protocols as payload)
- Open standard and proven Interoperability
- Query support for Real-time filtering of Events, Complex Event Processing, UDP, TCP
- Secure (SSL/TLS)
- Auto Data Endpoint changes discovered, with high Priority (QoS)
- Many to Many (tiered model), shards, partitioning, Domain based
- Persistence/Store (Cache, DB)
- Integrated collection and integrations beyond Telemetry for heterogeneous networks
- Flexible extensible framework through DDS enhance Agents and VNFs

RESP

- Common Event Stream, and beyond
- Reactive real-time Processing and Batch Processing
- Distributed
- Concurrent, HA, Secure
- Scalable end to end processing for edge, Lake
- Many to Many (tiered model), shards, partitioning, Domain based
- Persistence/Store (Cache, DB)
- Integrated collection and integrations beyond Telemetry for heterogeneous networks
- Flexible extensible framework through DDS enhance Agents and VNFs
## Core features

<table>
<thead>
<tr>
<th>Core features</th>
<th>DDS</th>
<th>Kafka</th>
<th>Kafka Streaming</th>
<th>ONAP Gaps for Distributed Streaming and Data Synchronization</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Memory Distributed DB</td>
<td>✓ Data/Object Centric DB Store, with Data Query Language support</td>
<td>Log based, Message Store, no visibility at the Data level</td>
<td>Query layer based on Message Store, no visibility at the Data level</td>
<td>Further DDS provides “Eventual Consistency” based data synchronization in Global Space providing consistent state</td>
</tr>
<tr>
<td>Integrated Efficient secure publish-subscribe binary-level communication</td>
<td>✓ Asynchronous Pub/Sub with distributed Domain based partitioning, data event Pub/Sub is strongly typed(binary) and state oriented</td>
<td>Text-based events, JSON based encoding, overhead communication and processing</td>
<td>Text-based events, JSON based encoding, overhead communication and processing</td>
<td>Further DDS DB cache and pub/sub are intelligent, as DDS tracks state of the data events stored. It allows for application to become model driven through well structure data models</td>
</tr>
<tr>
<td>System event reporting and logging</td>
<td>✓ Data/Object Centric DB Store, with Data Query Language support Rich Pub/Sub event tracking, reporting and logging with fine tuning through knobs for optimizing the data event traffic</td>
<td>Relies on TCP for flow control</td>
<td>Relies on TCP for flow control</td>
<td>DDS provides rich event tracking metrics for optimal data event traffic in the network on both Publisher and Subscriber end</td>
</tr>
</tbody>
</table>
## ONAP Gaps for Distributed Streaming and Data Synchronization

<table>
<thead>
<tr>
<th>Core Features</th>
<th>DDS</th>
<th>Kafka</th>
<th>Kafka Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Distributed authentication and access control</strong></td>
<td>✓ Distributed Authentication and Access control</td>
<td>✓ Authentication</td>
<td>DDS allows fine grain access and control Domain- &gt;Partition-&gt; QoS/Topic- &gt; Subs/Pubs</td>
</tr>
<tr>
<td><strong>Distributed scheduling and Processing management</strong></td>
<td>✓ Process management is present, DDS tracks the Publishers and Subscribers and re-syncs data upon recovery/re-start</td>
<td>Not present</td>
<td>Not present</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>DDS auto scheduling and re-start is an overlay, not present natively</strong></td>
</tr>
</tbody>
</table>
End to End Geo-Distributed System (RESP)

Achieve an Equilibrium State through a Model Driven Realtime event Stream interconnect

Synchronize Reliably

Clustered, Distributed, Reliable, HA, Cloud Native

System Engineered to be SLA compliant
Access control segregation for Global data share

Geo-Distributed Tiered Deployment
(inter-Carrier Use Case)
DCAE & RESP Integration

1. Data Streamed from southbound devices
2. VES collector receives RT stream & existing pull based measured data (acts as a proxy)
3. RT Stream can be directly received by DMaaS with Akka based routing
4. Microservices receive events and perform analytics processing using Flink through Beam APIs
5. Flink processing (Beam)
6. Microservices return processed result out of DCAE
7. As DMaaS Kafka/DDS event for Policy etc

Note: both existing and new DDS flows co-exist
ONAP RESP Logical Interface Flow

- RESP Logical Software building block tools
  - DDS, Akka, Beam & Flink
- Figure shows a flow sequence for RESP and some ONAP components
  a) A new end to end Realtime event streaming flow
  b) Current event flow can co-exist
  c) 2,3 internal RESP Routing and Processing
  d) 1,4,5,6 show external RT. and current event flow Close Loop
- VNF can push DDS based event stream
- Proxy can push DDS based event stream
DDS and Kafka

- Geo Distributed Global Data Centricity and Synchronization
- Low latency Pub/Sub P2P, Broker(optional) and N2M
- Push to Subscribers
- Query and Filter on data-value
- Data Centric
- Best effort, last-n, reliable and exactly once
- 20+ QoS settings

- Broker Based
- Pull from Cluster
- Opaque Data
- Message Centric
- At least once, at most once, exactly once
- Controlled reliability and durability
DDS Atomic loosely coupled Asynchronous Distributed Data Centric

The communication between the DataWriter and the DataReader can use UDP/IP (Unicast and Multicast) or TCP/IP
## DDS and Kafka Overview

<table>
<thead>
<tr>
<th></th>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scale</strong></td>
<td>Geo Distributed, data centric and Distributed synchronization, further through its Domain concept provides native tiered deployment capability</td>
<td>Clustered/centralized deployment</td>
<td>Distributed large scale topic based classification and eventual consistent synchronization</td>
<td>Support through a hub/spoke model</td>
</tr>
<tr>
<td><strong>Data Durability</strong></td>
<td>Eventual Consistency based Replicated and inbuilt Cache and Persistent DB</td>
<td>Persistence supported through Logs on Disk</td>
<td>Data Durability is fine grained controlled through QoS settings</td>
<td>Log based</td>
</tr>
<tr>
<td></td>
<td>Peer DDS DataWriter and DDS DataReader Cache and persistent Cache support</td>
<td>Log/file based persistence support</td>
<td>Very flexible Cache for Data providing variable volatile, transient and persistent cache, native wire protocol support for high performance processing</td>
<td>Persistence based log files, file handle becomes a challenge for large deployments, envelope, payload needs to be further extracted and processed</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Purpose built for real-time messaging through QoS configuration, ~ 35-50 µsec*</td>
<td>Low-latency achieved only by reducing replication/reliability parameters</td>
<td>DDS provides low latency times, by default UDP based with native reliability through built in wire protocol, native multicast support also facilitates pointed speedy delivery. Tracks and reports delivery metrics for tuning.</td>
<td>Low latency available at the risk of reducing HA/replication in the centralized broker deployment</td>
</tr>
</tbody>
</table>
# DDS and Kafka Overview

<table>
<thead>
<tr>
<th>QoS</th>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Very rich QoS service features, with 20+ features.</td>
<td>No QoS, or guarantee of the data except through TCP</td>
<td>Very Rich and flexible QoS allows for a global distributed Data Space to be created between networking nodes/elements, EMS systems and other applications (P2P or P2Centralized)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication</th>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poll, Asynchronous, Synchronous (UDP, TCP, multicast)</td>
<td>Asynchronous, Synchronous (TCP)</td>
<td>Lightweight UDP based and reduce chatter through multicast support, flow control built-in</td>
<td>Uses TCP, high throughput support</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strong Data Type support</th>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Data type support and IDL based code bindings for consistency in programming languages (C, C++, Java..)</td>
<td>Message Oriented, data encapsulated as payload</td>
<td>DDS is Data centric through a standard way of IDL or Protocol Buffers, allowing polyglot support</td>
<td>Only Message level handling, and data needs to be de-serialized at the application</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Auto Discovery</th>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Discovery of Participants and Topic</td>
<td>N/A</td>
<td>This allows for very efficient deployment model, shields from topology changes and provide a plug-play approach</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## DDS and Kafka Overview

<table>
<thead>
<tr>
<th>DDS</th>
<th>Kafka</th>
<th>DDS Notes</th>
<th>Kafka Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query &amp; Filter</strong></td>
<td>DDS supports Content based Query and filtering on Data, support both Queue and Data Cache functionality</td>
<td>No support</td>
<td>Further, flexible to even provide the state of each data sample e.g if the data was read and what time the data was read</td>
</tr>
<tr>
<td><strong>HA and Resiliency</strong></td>
<td>DDS provides Geo distributed HA and Resiliency</td>
<td>Distributed, HA</td>
<td>QoS settings, facilitate replication of data upon node failures and re-starts also data is consistently synced between pub/sub peers</td>
</tr>
<tr>
<td><strong>Version management and release updates</strong></td>
<td>With x-types or Protocol Buffers can support dynamic insertion of new messages and version management capability</td>
<td>Not available</td>
<td>With IDL and Protocol Buffers, can achieve backward compatibility across pub/sub topics \ Any kind of version tracking would be an overlay implemented in the application</td>
</tr>
<tr>
<td><strong>Data Streaming</strong></td>
<td>Provides fine-grained control with QoS based ‘exactly once’ message delivery Pub/Sub and back pressure handling</td>
<td>Source only ‘exactly once’ capability</td>
<td>DDS is Data centric, allowing for flexibility in adapting to a heterogeneous network/product/service needs natively, one to one mapping to endpoint</td>
</tr>
</tbody>
</table>
ONAP Architecture + RESP

![ONAP Architecture Diagram]

- **VNF SDK**
  - Change Management Design
  - Design Test & Certification

- **CLAMP**
  - Catalog

**DESIGN-TIME (SDC)**
- Resource Onboarding
- Service & Product Design
- Policy Creation & Validation

**EXTERNAL SYSTEMS**
- Hypervisor / OS Layer
  - OpenStack
  - VMware
  - Azure
  - K8S
  - RackSpace

- Network Function Layer
  - VNFS
  - ... PNFs

- Managed Environment
  - Private Edge Cloud
  - MPLS
  - Private DC Cloud
  - IP
  - Public Cloud

**RUN-TIME**
- Policy Framework
- DCAE
- Service Orchestration Project
- A&AI/ESR
- Common Services
  - AAF
  - OOF
  - Logging
  - MUSIC
  - Others

**EXTERNAL SYSTEMS**
- 3rd Party Controllers
- sVNF, EMS

**ONAP External APIs**
- External Gateway
- OSS / BSS
- ONAP CLI
- U-UI
- ONAP Portal

**RELATED ONAP OPERATIONS**
- ONAP Operations Manager
- Integration
- VNF Requirements
- VNF Validation Program

**NOTES**
- Recipe/Eng Rules & Policy Distribution
  - Note 1 - VF-C is ETSI-aligned.
## DDS vs. Others

<table>
<thead>
<tr>
<th>Connector</th>
<th>Real Time</th>
<th>Exactly once</th>
<th>Durability</th>
<th>Storage Capacity</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Years</td>
<td></td>
</tr>
<tr>
<td>Kafka</td>
<td>Yes</td>
<td>Source only</td>
<td>Yes* (Flushed but not synced)</td>
<td>Days</td>
<td>Writes are replicated but may not persist to durable media. (flush.messages=1 bounds this but is not recommended)</td>
</tr>
<tr>
<td>RabbitMQ</td>
<td>Yes</td>
<td>Source only</td>
<td>Yes* (slowly)</td>
<td>Days</td>
<td>Durability can be added with a performance hit</td>
</tr>
<tr>
<td>Cassandra</td>
<td>No</td>
<td>Yes* (If updates are idempotent)</td>
<td>Yes</td>
<td>Years</td>
<td>App developers need to write custom logic to handle duplicate writes.</td>
</tr>
<tr>
<td>Sockets</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>DDS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>High-Performance distributed durability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Built-in support for circuit-breakers as well as batching</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Content awareness (filters and queries natively supported)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dynamic Discovery facilitates deployment and shields from topology changes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Support for IP multicast give unparalleled scalability</td>
</tr>
</tbody>
</table>
Mapping to OSI layers

User

App App ...

App

L7: Application
L6: Presentation
L5: Session
L4: Transport
L3: Network
L2: Data Link
L1: Physical

C/C++, Java, .Net, JavaScript, Python, etc.
DCPS
RPC
Security
X-Types
DDSI-RTPS
TCP
UDP
IP

802.3 802.11 [4] 802.1 ...

} DDS
Q1: What are the End point Use Cases I & II?
A1: Use Case I is a simple smart end point scenario, showing streaming interconnect between publishers (producers) and subscribers (consumers). Use Case II, however, is a full streaming, collection and analytics use case. Representing an end-to-end real-time data pipeline for streaming via RESP proposed solution.

Q2: Why asynchronous communication?
A2: The DDS middleware handles asynchronous communications via a backpressure mechanism coupled with the DDS QoS feature configurations. This takes the burden off the application code for handling any retries by enabling an end-to-end RESP-based data processing pipeline with backpressure capability (DDS<->Akka<->Beam + Flink).

Q3: What do you mean by real-time?
A3: The DDS middleware enables end-to-end real-time pub/sub streaming for data-centric events from wire protocol to middleware through DDS QoS feature settings (the QoS profile can be configured by taking into account the requirements of the user data (either real-time or near real-time).

Q4: Does DDS support the WAN traffic?
A4: Yes, it supports it very effectively, via the concept of “DDS Links”.

Q5: Does the existing DCAE flow change?
A5: No, this is complementary to the existing ONAP DCAE flows. Design-time and run-time enhancements are made to meet the RESP functional requirements (S7,S11)

Q6: Message guaranteed delivery; how is this handled?
A6: DDS with proper QoS configuration ensures that any lost event to be delivered due to the “Eventual Consistency” capability.

Q7: Does VNF need to be updated?
A7: The VES collector is updated to handle DDS-based real-time streaming (both southbound and northbound). On the southbound a VNF (a sensor, etc) can push data/measurements to the collector which in turn uses DDS to generate a real-time stream to be processed by the analytics engine.