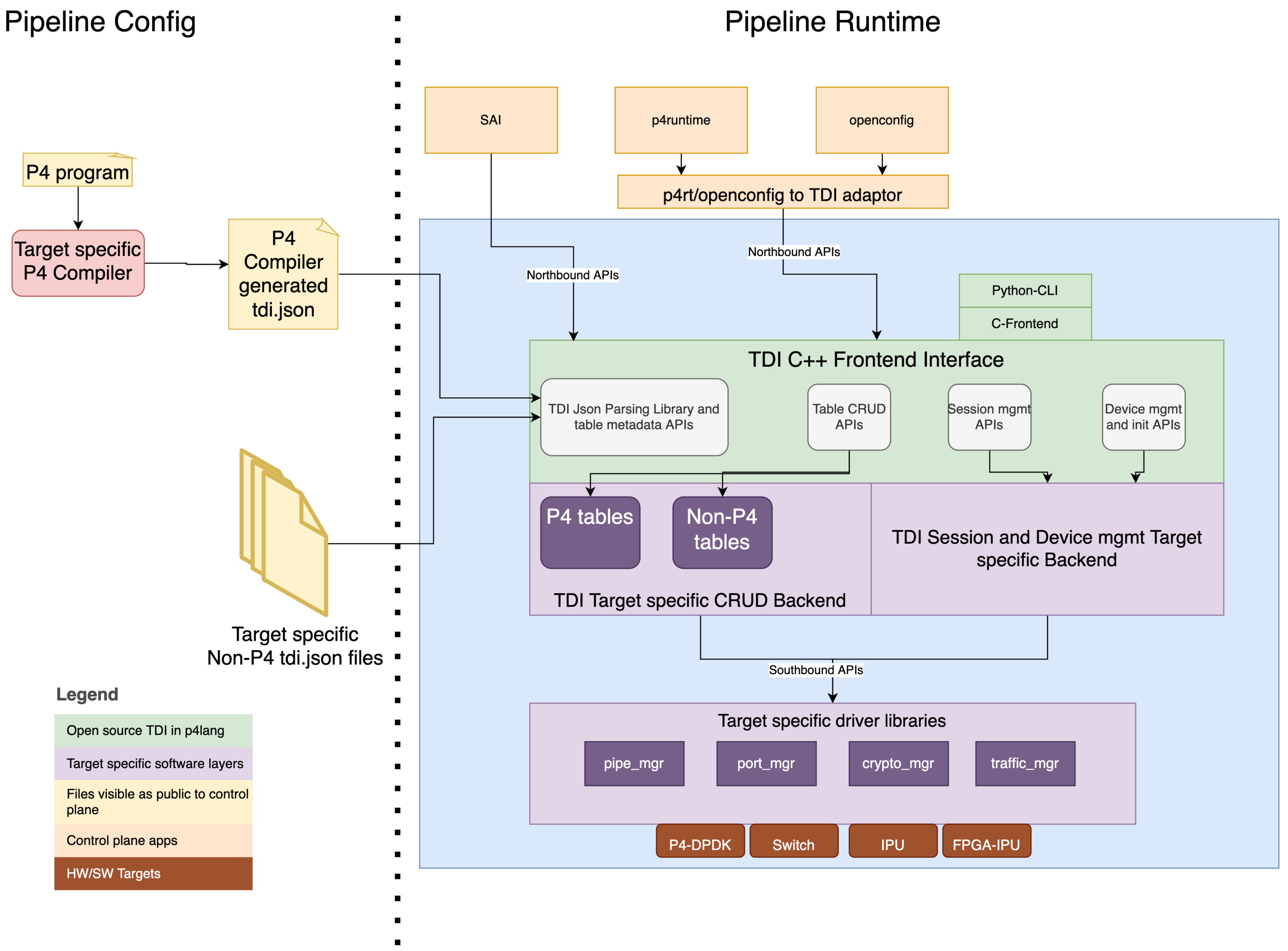
Table Driven Interface (TDI): Usages and Advantages

Sayan Bandyopadhyay
James Choi

What is TDI (Table Driven Interface) ?

- Table Driven Interface (TDI) is a Target Agnostic Interface. Present under p4lang/tdi
- A common frontend for TDI exists in open-source which can be used by control plane applications.
- Targets need to implement their backend for providing support.
- Every runtime entity is represented as one or multiple tables whether P4(MatchAction, ActionProfile) or non-P4 (Port configuration)
- tdi.json is a json-based contract between TDI frontend and control plane on how these tables look like. Similar to p4info in p4runtime
- Compiler generates a tdi.json for P4 entities.
  Target drivers use target-specific handwritten tdi.json files for fixed features
- TDI json parsing library present in p4lang/tdi is used to parse all the json files and bringup TDI module for the target.


Pipeline Config
Pipeline Runtime
P4 program
Target specific P4 Compiler
P4 Compiler generated tdi.json
Target specific Non-P4 tdi.json files
SAI
p4runtime
openconfig
p4rt/openconfig to TDI adaptor
Northbound APIs
Northbound APIs
Python-CLI
C-Frontend
TDI C++ Frontend Interface
TDI Json Parsing Library and table metadata APIs
Table CRUD APIs
Session mgmt APIs
Device mgmt and init APIs
P4 tables
Non-P4 tables
TDI Target specific CRUD Backend
TDI Session and Device mgmt Target specific Backend
Southbound APIs
Target specific driver libraries
pipe_mgr
port_mgr
crypto_mgr
traffic_mgr
P4-DPDK
Switch
IPU
FPGA-IPU
Legend
Open source TDI in p4lang
Target specific software layers
Files visible as public to control plane
Control plane apps
HW/SW Targets

P4 extern and non-P4 features to json structure

- A Match-Action Table (MAT) is converted to a tdi.json table as shown.
- Tdi.json table broadly contains
    - Key : Multiple key fields
    - Data
        - Action-specific data fields
        - "Common" data fields like counter data

- Equivalent C-like struct and union construct can be realized using key, actions, data. This is used to translate a non-P4 feature (fixed feature) like Ports to a TDI table

```
DirectCounter<bit<32>> CounterType_t.PACKETS_AND_BYTES) counter;
action hit(bit<9> port){
 ig_md.ucast_egress_port = port;
}
action miss(bit<1> drop){
 ig_dprsr_md.drop_ctl = drop;
}
table forward{
 key = { hdr.ethernet.dst_addr : exact;
         hdr.ethernet.src_addr : ternary;}
 actions = { hit; @defaultonly miss; }
 counters = counter;
 size = 1024;
}
```

```
struct action_hit { port;}
struct action_miss { drop;}
struct counter_t {PACKET_COUNT; BYTE_COUNT}
struct key_t {
 hdr.ethernet.dst_addr;
 hdr.ethernet.src_addr;
};
struct data_t {
 union {
  struct action_hit a1;
  struct action_miss a2;
 } action;
 struct counter_t c1;
};

struct entry_t {
 struct key_t key;
 struct data_t data;
};
```

```
{ "name" : "pipe.SwitchIngress.forward",
  "id" : 37882547,
  "table_type" : "MatchAction_Direct",
  "size" : 1024,
  "depends_on" : [],
  "key" : [
    { "id" : 1,
      "name" : "hdr.ethernet.dst_addr",
      "match_type" : "Exact",
      "type" : {"type" : "bytes", "width" : 48}
    },
    { "id" : 2,
      "name" : "hdr.ethernet.src_addr",
      "match_type" : "Ternary",
      "type" : {"type" : "bytes", "width" : 48}
    }
  ],
  "action_specs" : [
    { "id" : 32848556,
      "name" : "SwitchIngress.hit",
      "action_scope" : "TableAndDefault",
      "data" : [
        { "id" : 1,
          "name" : "port",
          "type" : {"type" : "bytes", "width" : 9}
        }
      ]
    },
    { "id" : 17988458,
      "name" : "SwitchIngress.miss",
      "action_scope" : "DefaultOnly",
      "data" : [
        { "id" : 1,
          "name" : "drop",
          "type" : {"type" : "bytes", "width" : 1}
        }
      ]
    }
  ],
  "data" : [
   {  "id" : 65553,
      "name" : "$COUNTER_SPEC_BYTES",
      "type" : {"type" : "uint64", "default_value" : 0}
   },
   {  "id" : 65554,
      "name" : "$COUNTER_SPEC_PACKETS",
      "type" : {"type" : "uint64", "default_value" : 0}
   },
  ],
  "supported_operations" : [],
  "attributes" : ["EntryScope"]
},
```

TDI json table structure to API mapping

```
{ "name" : "pipe.SwitchIngress.forward",
  "id" : 37882547,
  "table_type" : "MatchAction_Direct",
  "size" : 1024,
  "depends_on" : [],
  "key" : [
    { "id" : 1,
      "name" : "hdr.ethernet.dst_addr",
      "match_type" : "Exact",
      "type" : {"type" : "bytes", "width" : 48}
    },
    { "id" : 2,
      "name" : "hdr.ethernet.src_addr",
      "match_type" : "Ternary",
      "type" : {"type" : "bytes", "width" : 48}
    },
    { "id" : 3,
      "name" : "$MATCH_PRIORITY",
      "match_type" : "Exact",
      "type" : {"type" : "uint32"}
    }
  ],
  "action_specs" : [
    { "id" : 32848556,
      "name" : "SwitchIngress.hit",
      "action_scope" : "TableAndDefault",
      "data" : [
        { "id" : 1,
          "name" : "port",
          "type" : {"type" : "bytes", "width" : 9}
        }
      ]
    },
    { "id" : 17988458,
      "name" : "SwitchIngress.miss",
      "action_scope" : "DefaultOnly",
      "data" : [
        { "id" : 1,
          "name" : "drop",
          "type" : {"type" : "bytes", "width" : 1}
        }
      ]
    }
  ],
  "data" : [
   {  "id" : 65553,
      "name" : "$COUNTER_SPEC_BYTES",
      "type" : {"type" : "uint64", "default_value" : 0}
   },
   {  "id" : 65554,
      "name" : "$COUNTER_SPEC_PACKETS",
      "type" : {"type" : "uint64", "default_value" : 0}
   },
  ],
  "supported_operations" : ["CountersSync"],
  "attributes" : ["EntryScope"]
},
```

Table

	Key	Data: Action	Data: Common Data

Entry 1	dst_addr = "AA:BB:CC:DD:EE:FF"
	src_addr = "11:22:33:44:55:66"
	src_addr_mask = "0xFFFFFFFFFFFF"
	$MATCH_PRIORITY = 1
		(action = hit)
		port = "3"
			Packets = 0
			Bytes = 0

Entry 2	dst_addr = "AA:BB:CC:DD:EE:EE"
	src_addr = "11:22:33:44:55:66"
	src_add_mask = "0xFF00"
	$MATCH_PRIORITY = 5
		(action = hit)
		port = "6"
			Packets = 10
			Bytes = 500

Entry 3	dst_addr = "AA:BB:CC:DD:EE:EE"
	src_addr = "11:22:33:44:55:66"
	src_add_mask = "0xFFFF"
	$MATCH_PRIORITY = 2
		(action = hit)
		port = 7
			Packets = 30
			Bytes = 2000

Default Entry	■	(action = miss)
		drop = 1
			Packets = 0
			Bytes = 0

Table APIs

Per-entry APIs

1. entryAdd
2. entryDel
3. entryMod
4. entryGet
5. entryGetFirst
6. entryGetNextN

Table-wide APIs

1. clear
2. attributeSet
3. operationsExecute

Default Entry APIs

1. defaultEntrySet
2. defaultEntryReset
3. defaultEntryGet

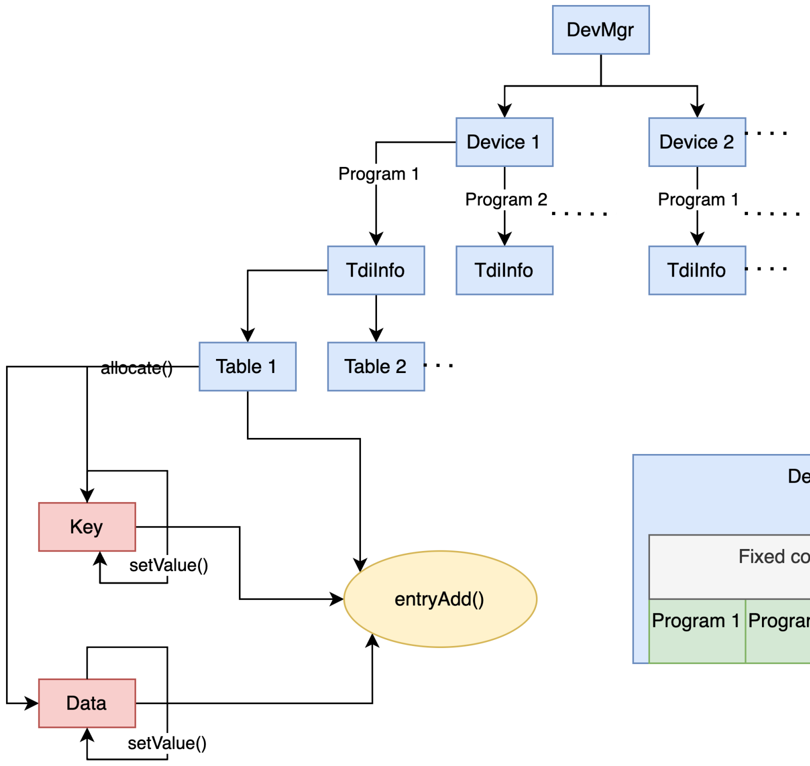
API types and a simple API workflow

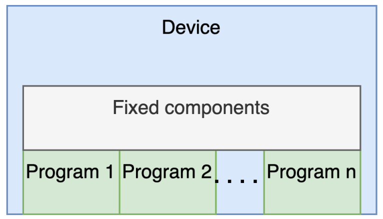
1. Infrastructure APIs
    1. Device Management
    2. Table Metadata
2. Session APIs : Batching, transaction APIs
3. Table APIs

Device

- A Device object can be configured with multiple programs depending upon pipeline split supported
- Every program has one TdiInfo object which in turn has tdi::Table objects.
- Fixed functions can either be present on the device itself, or they are all accessible through each of the program specific TdiInfo objects.

```
const tdi::Device *device;
DevMgr::DeviceGet(tdi_dev_id_t dev_id, &device);

target = device->createTarget(); // Target object to specify target
                                 // specific subdevice or group of
                                 // subdevice like a pipe_id or pipe_all
session = device->createSession();// Session object to control parallism
                                 // across threads

const tdi::TdiInfo *tdi_info;
device->tdiInfoGet(string program_name, &tdi_info);

const tdi::Table *table;
tdi_info->tableGet(string name, &table);


unique_ptr<tdi::TableKey> key;
table->keyAllocate(&key);

unique_ptr<tdi::TableData> data;
table->dataAllocate(action_id = 1,      // Action ID of hit
                        &data);


key->setValue(1,                                  // field_id of dst_addr
  tdi::KeyFieldValueExact(0x11223344));           // value
key->setValue(2,                                  // field_id of src_addr
  tdi::KeyFieldValueTernary(0x11223344,           // value and mask
    0xffffff));

data->setValue(1,       // field_id of data field = port
        4);             // value


table->entryAdd(target, session, key, data);
```

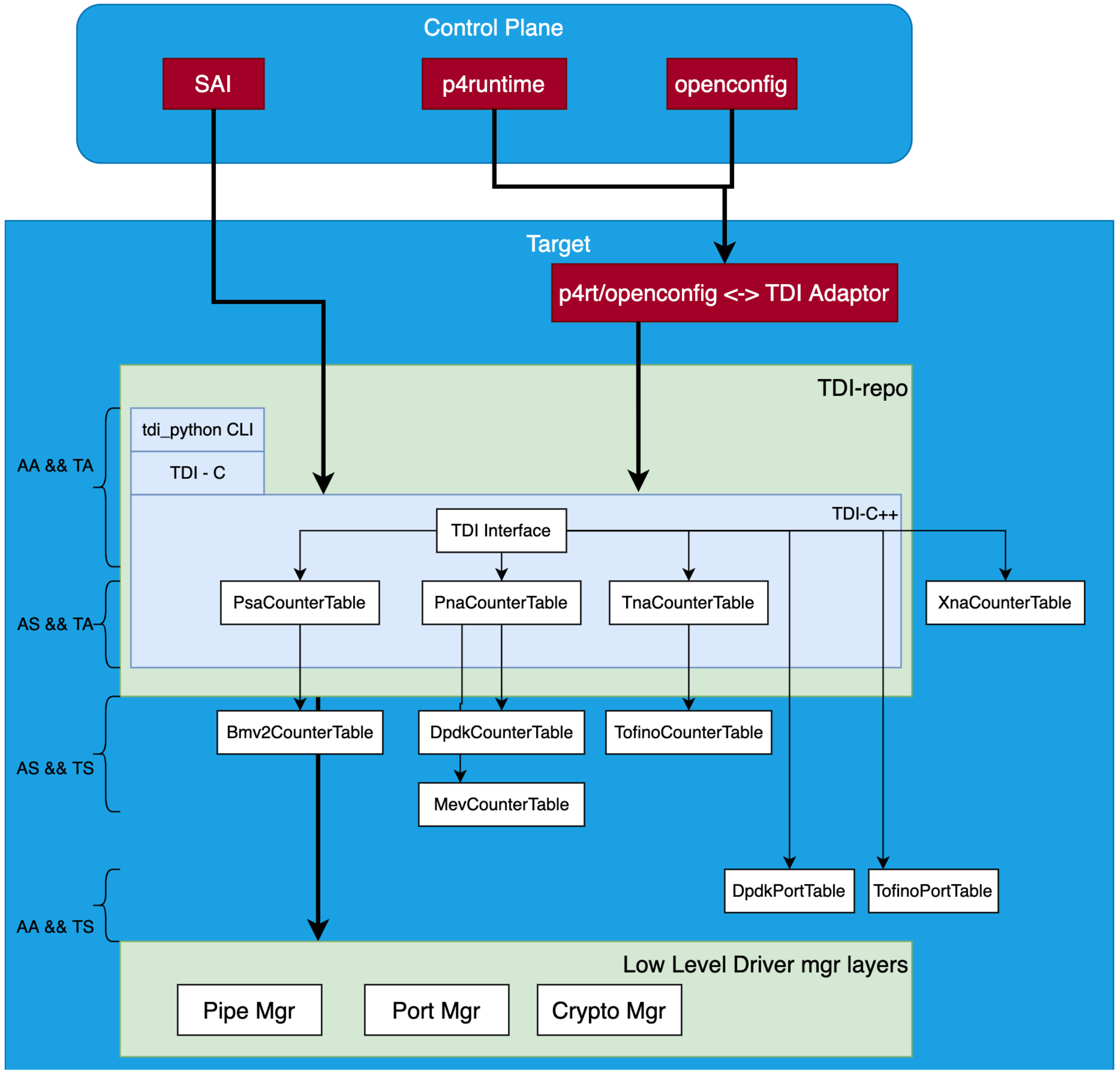




TDI Frontends

- Different TDI interfaces
  1. C++ frontend
  2. C frontend
  3. Python CLI
- Uses C++ inheritance to achieve abstraction and usage of common code for any P4 architecture
- Any new P4 architecture doesn't need to add code here in the p4lang tdi repo but it can be added here to maintain common code for certain externs of a P4 architecture
- README to TDI : [https://github.com/p4lang/tdi/blob/main/README.md]

- AA : Architecture (P4) Agnostic
- AS : Architecture (P4) Specific
- TA: Target Agnostic
- TS: Target Specific



Advantages of TDI

1. **Control plane code uniformity**: Since the frontend is target and P4 arch agnostic, it can be used by control planes to write applications which are uniform across targets.
2. **Target vendor ease of feature addition**: The frontend and backend split allows device vendors to provide support for P4 and non-P4 features easily in a consistent manner.
3. **No extensions needed for new features**: The C++ frontend interface and the TDI tools (python CLI, C-Frontend) themselves are all consistent and don't require any change when support for a new P4-extern or Non-P4 feature needs to be added for a target

TDI vs P4Runtime

TDI	P4Runtime
• Primarily Native APIs in C++ and C	• Primarily gRPC and protobuf based APIs
• TDI APIs are based on an abstract structure called a "table". API flow to program any table is the same. Match tables, Counters, Registers are all tables.	• API flow to program different entities are different
• Provides an interface which is P4 Architecture agnostic. Can potentially be used for non-networking API usage as well like storage.	• Tightly knit with PSA. There is provision for p4runtime to work with non-PSA externs but the natively supported externs are PSA-specific like DirectMeter, DirectCounter etc.
• TDI is extern independent. Any new extern addition in any P4 architecture, doesn't require frontend changes.	• P4Runtime core APIs and proto do require additional changes. However, the same could be achieved via "Externs" in P4Runtime, but since the message structure in it itself is completely dependent on the vendor, it doesn't have a generic specification guideline.

A TDI table needs to be added to the json (equivalent to the P4info in p4runtime).
The runtime itself doesn't need any new compilation or new messaging

In order to add support for extensions in P4runtime , new target-specific protobuf messages are required which add compile time and target-specific dependencies.

```
message P4Ids {
  enum Prefix {
    UNSPECIFIED = 0;
    MY_NEW_PACKET_COUNTER = 0x81;
  }
}
```

P4info message

```
message MyNewPacketCounter {
  // corresponds to the T type parameter in the P4 extern definition
  p4.config.v1.P4DataTypeSpec type_spec = 1;
  // constructor argument
  int64 size = 2;
}
```

P4runtime message

```
// This message enables reading
// index
message MyNewPacketCounter {
  int64 index = 1;
  p4.v1.P4Data data = 2;
}
```

What is P4-DPDK-TARGET?

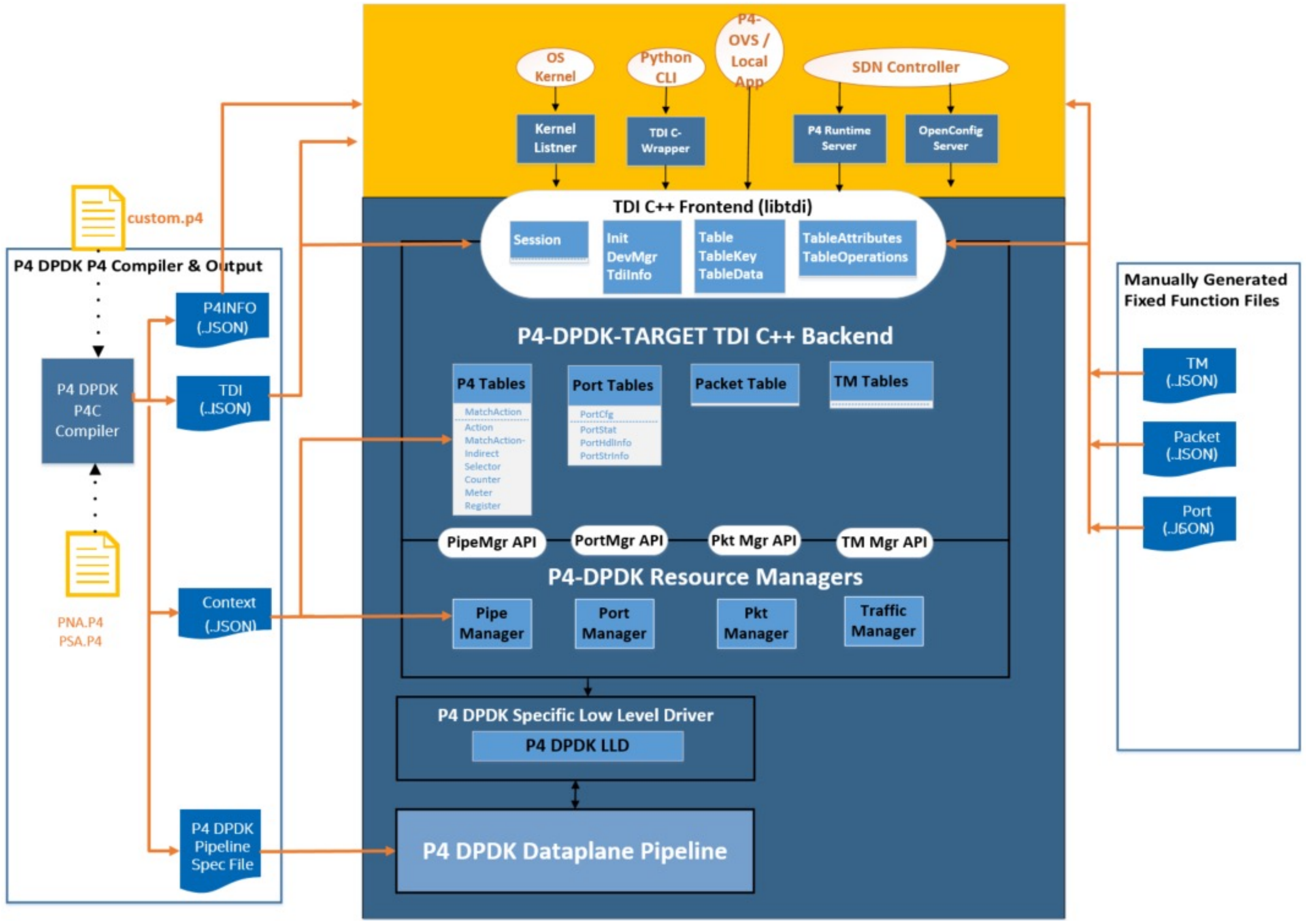
- P4-DPDK-TARGET a p4lang project for the enabling management of P4 DPDK pipeline through TDI.
  - https://github.com/p4lang/p4-dpdk-target
  - Upstreaming of TDI implementation is in progress. Currently contains code for BFRT interface, which is predecessor to TDI.
- P4 DPDK pipeline is a DPDK based SW pipeline that is configurable through P4.
  - https://github.com/DPDK/dpdk/tree/main/lib/pipeline/rte_swx_*.*
  - Supports P4 PNA & PSA architectures
  - Runtime files generated by P4 compiler with DPDK backend.
  - https://github.com/p4lang/p4c/tree/main/backends/dpdk

- TDI frontend includes:
  - TDI base C++ class declarations
  - TDI.json parser that support the current TDI.JSON schema.
  - TDI CLI
- P4-DPDK-TARGET backend contains:
  - TDI C++ backend for P4 tables for PNA & PSA
  - TDI C++ backend for non-P4 table.
    - Currently supports port/vport related tables
  - P4 DPDK specific resource managers
  - Low-level driver (LLD) for P4 DPDK pipeline API
  - Sample application (bf_switchd)
  - Examples
  - https://github.com/p4lang/p4-dpdk-target#readme for more info.

How does P4-DPDK-TARGET implement TDI?

TDI implementation layers

- TDI C++ base class frontend
  - Referenced through submodule to TDI
- TDI C++ class P4 DPDK backend
  - Map TDI table key & data fields to bitarray based on target specific context.json
  - Table specific implementation
- P4 DPDK target-specific resource manager libraries
  - Pipe Manager
    - Southbound interface for TDI P4 backend
    - Session implementation
  - Port Manager
    - Southbound interface for Port/Vport backend
- P4 DPDK LLD
  - Simple target access layer for P4-DPDK target



What device config does P4-DPDK-TARGET support?

Device Configurations Supported

TDI Infra Objects	Supported by TDI	Supported in P4-DPDK-TARGET
Device	Multiple devices per TDI library	Single device per TDI library
ProgramConfig (P4 program & TDI.JSON)	Multiple ProgramConfig per device	Upto 4 ProgramConfig per device
P4Pipeline (Target binary & CONTEXT.JSON)	Multiple P4Pipeline per ProgramConfig	One P4Pipeline per ProgramConfig
PipeScope (Grouping of target pipelines in device)	Multiple logical pipes per P4Pipeline	Single logical pipe per P4Pipeline

```
# Config file used by bf_switchd application
   {
"p4_devices": [
         {
            "device-id": 0,
            "p4_programs": [
               {
                  "program-name": "<P4 program name>",
                  "tdi-config": "share/dpdk/<P4 program name>/tdi.json",
                  "port-config": "share/dpdk/<P4 program name>/ports_topology.json",
                  "p4_pipelines": [
                     {
                        "p4_pipeline_name": "pipe",
                        "context": "share/dpdk/<P4 program name>/pipe/context.json",
                        "config": "share/dpdk/<P4 program name>/pipe/<P4 program
name>.spec",
                        "path": "share/dpdk/<P4 program name>“
                        “pipe_scope”: [0, 1, 2, 3]
                     }
                  ]
               }
            ],
         }
      ]
```

How is P4Runtime supported?

- P4 function characteristics:
  - Programmable with P4
  - TDI JSON files are compiler generated and context.json is used for mapping to target pipeline.
  - P4 function resources are allocated at P4Pipeline level by compiler and managed by driver .
  - P4 function JSON files can be loaded at TDI library initialization time and at runtime thru setforwardingpipelineconfig
- Implement tdi::Table C++ classes for P4Runtime entities
  - Each P4Runtime message handled by specific TDI backend C++ class for specific table type
  - TableType in TDI.JSON can be used to differentiate tables by applications
- Integrated with ONF Stratum as part of IPDK integration.
  - https://github.com/stratum/stratum
  - IPDK Container available at: https://github.com/ipdk-io/ipdk/blob/main/build/networking/README_DOCKER.md

P4Runtime Messages	P4C TDI.JSON TableType	P4-DPDK-TARGET C++ Backend
• TableEntry w/ Action • DirectCounterEntry • DirectMeterEntry	• MatchAction_Direct	• MatchActionDirect
• TableEntry w/ action profile member or group	• MatchAction_Indirect • MatchAction_Indirect_Selector	• MatchActionIndirect
• ActionProfileGroup	• Selector	• Selector
• ActionProfileMember	• Action	• Action
• CounterEntry	• Counter	• CounterIndirect
• MeterEntry	• Meter	• MeterIndirect
• RegisterEntry	• Register	• RegisterIndirect

How is OpenConfig supported?

- Fixed function characteristics:
    - Not programmable with P4
    - TDI fixed function JSON files are hand-written and do not use context.json for mapping to target pipeline.
    - Fixed function resources are managed at device level and shared by all P4 pipelines.
    - Fixed function JSON files are loaded at TDI library initialization time, not at runtime thru setforwardingpipelineconfig
- Integrated with ONF Stratum as part of IPDK integration.
    - Private OpenConfig YANG model for port/vport is used.
- Set of supported fixed functions
    - Port and vport mgmt
    - https://github.com/p4lang/p4-dpdk-target/blob/main/src/bf_rt/bf_rt_port/dpdk/bf_rt_port.json (*TDI port implementation is under development.)

OpenConfig Model	**TDI Table**	**TDI.JSON TableType**
openconfig-interfaces-stratum.yang	PortCfgTable	PortConfigure
openconfig-interfaces-stratum.yang	PortStatTable	PortStat

What is the roadmap?

TDI & P4-DPDK-TARGET Roadmap

1. TDI specification document v1.0 out by July’22
2. TDI integrated with P4-DPDK by July’22
    1. Currently working on upstreaming TDI P4-DPDK backend code.
3. TDI WG integration with p4Lang by July’22 (tentative)

Thank You

Additional Resources

1. "Table Driven Interface API Opens P4-Programmable Data Plane Features", Sunil Ahlwalia, Intel, blog, https://opennetworking.org/news-and-events/blog/table-driven-interface-api-opens-p4-programmable-data-plane-features/

2. P4 TDI GitHub: https://github.com/p4lang/tdi

3. Infrastructure Programmer Development Kit (IPDK) website: https://ipdk.io/

How is TDI table implemented?

```
class tdi::Table {
 public:
  virtual tdi_status_t entryAdd(const tdi::Session &session,
                                const tdi::Target &dev_tgt,
                                const tdi::Flags &flags,
                                const tdi::TableKey &key,
                                const tdi::TableData &data) const;
 protected:
 Table(const TdiInfo *tdi_info, const TableInfo *table_info)
     : tdi_info_(tdi_info), table_info_(table_info){};
}
```

```
class MatchActionDirect : public tdi::Table {
 public:
  MatchActionDirect(const tdi::TdiInfo *tdi_info,
                    const tdi::TableInfo *table_info)
      : tdi::Table( tdi_info, table_info) {}

 virtual tdi_status_t entryAdd(const tdi::Session &session,
                               const tdi::Target &dev_tgt,
                               const tdi::Flags &flags,
                               const tdi::TableKey &key,
                               const tdi::TableData &data) const override;
 private:
  // Table specific private members
  std::map<tdi_id_t, bool> act_uses_dir_meter;
  std::map<tdi_id_t, bool> act_uses_dir_cntr;
  std::map<tdi_id_t, bool> act_uses_dir_reg;
};
```