SUMMER OF CODES 2016

HTTPS://XGITLAB.CELS.ANL.GOV/CODES/SOC-2016-WHATSNEWINCODES

Argonne NATIONAL LABORATORY

WHAT'S NEW IN CODES

JOHN JENKINS
Argonne National Laboratory
jenkins@mcs.anl.gov

2016-07-13
Argonne National Laboratory

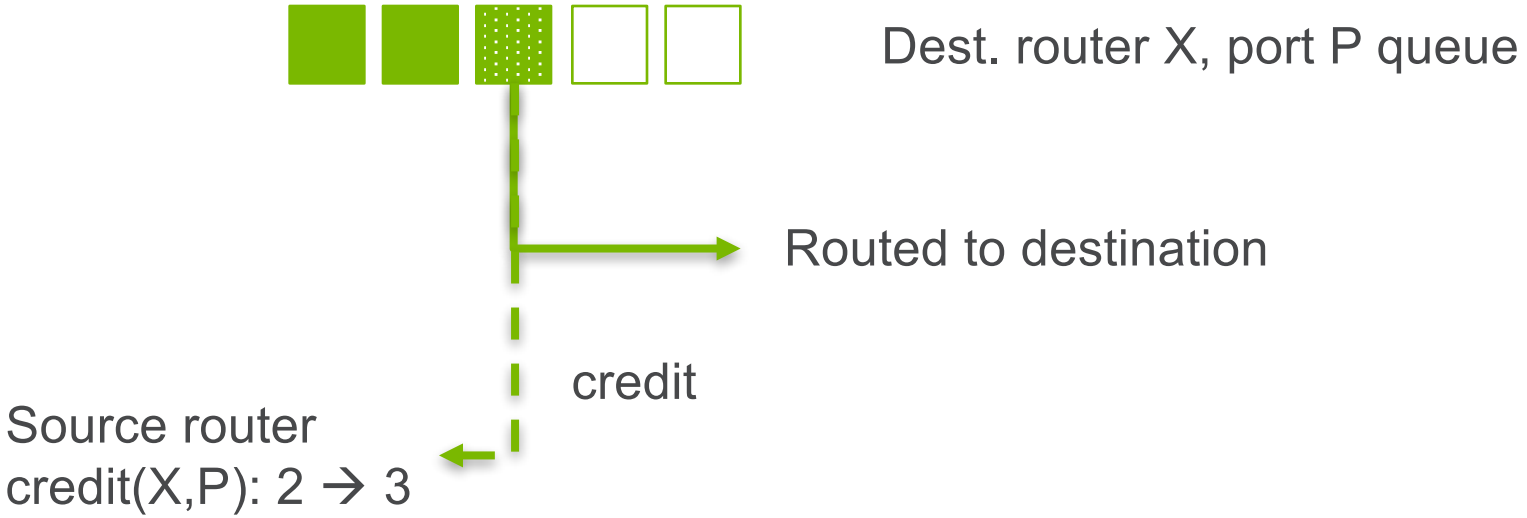
OVERVIEW

- Refined Networking Models
  - Credit flow control (torus, dragonfly)
  - Adaptive routing (dragonfly)
- Slim Fly network topology – covered in Noah's talk
- New and updated APIs
  - Message passing
  - LP mapping
  - Workloads
  - Modelnet state sampling
- Full list in CODES repository – doc/RELEASE_NOTES

REFINED NETWORKING MODELS

CREDIT-BASED FLOW CONTROL IN DRAGONFLY/TORUS NETWORKS

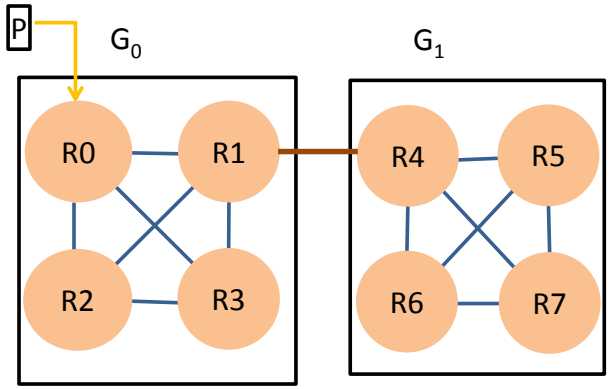
- Before – buffer overflow on routers killed the simulation
- After – credit-based flow control
  - Source nodes/routers track buffer availability on destination nodes/routers



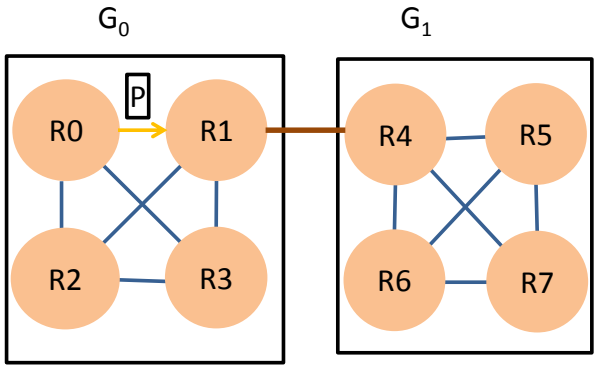
DRAGONFLY ADAPTIVE ROUTING

- Minimal and random non-minimal (Valiant) routing available
- Adaptive routing - minimal routing w/ random non-minimal fallback on full destination queue

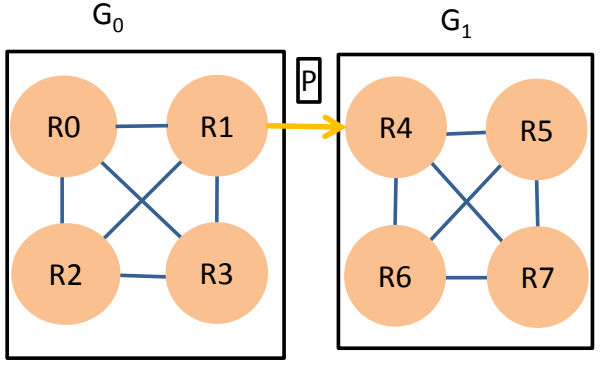
Non-minimal – repeat (ii), (iii) to intermediate group



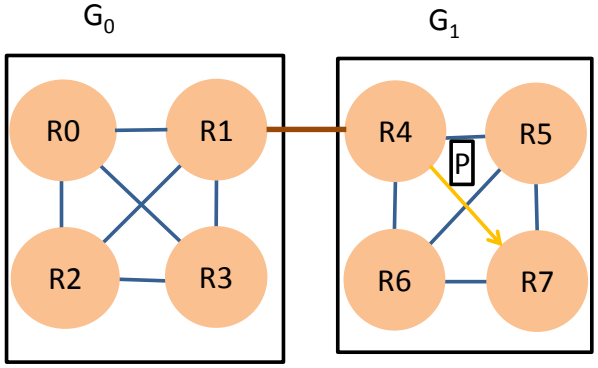
(i) Packet arrives at R0, Destination Router = R7



(ii) Packet traverses to R1 over local channel



(iii) Packet traverses from R1 to R4 over the global channel



(iv) Packet traverses to R7 over local channel

NEW AND UPDATED APIS

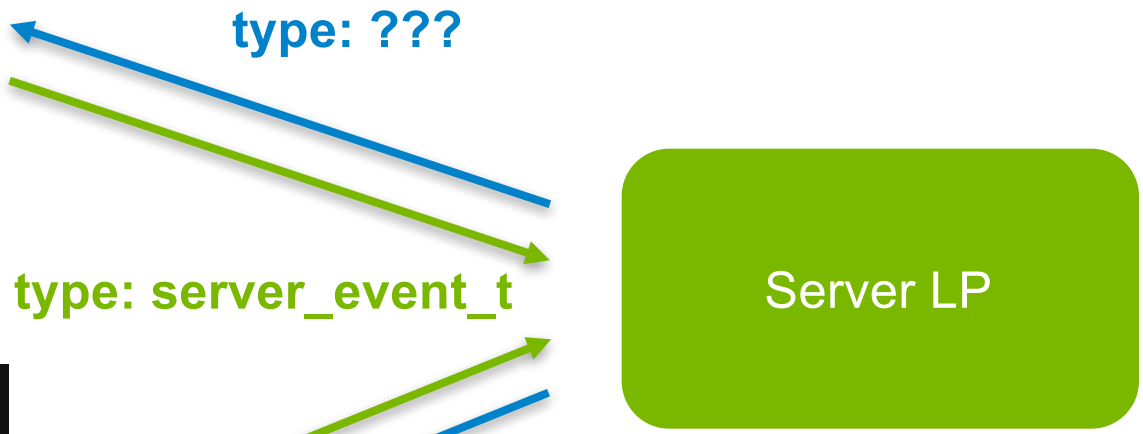
CALLBACK API

Convention for RPC-style event management

- Problem: event destination data layout needs to be known at the caller, makes it awkward to deal with multiple LP types

```
struct client_msg
{
    msg_header h;
    int dummy[5];
    int tag;
    server_return_t ret;
};
```

```
struct client2_msg
{
    int dummy;
    int tag;
    server_return_t ret;
    msg_header h;
};
```



CALLBACK API

Convention for RPC-style event management

- Solution
  - define convention for multi-LP-type control flow (RPC-flavored)
    - Assume: callee event type known
    - Assume: caller event type unknown
  - provide helper datatypes, functions/macros to simplify data structure management

- Convention
  - msg_header (codes/lp-msg.h)
    - header identifying msg type
    - provided by caller
  - tag
    - differentiate concurrent messages from same source LP
    - provided by caller
  - callee ”return” type
    - pass data from callee to caller
    - filled in by callee.

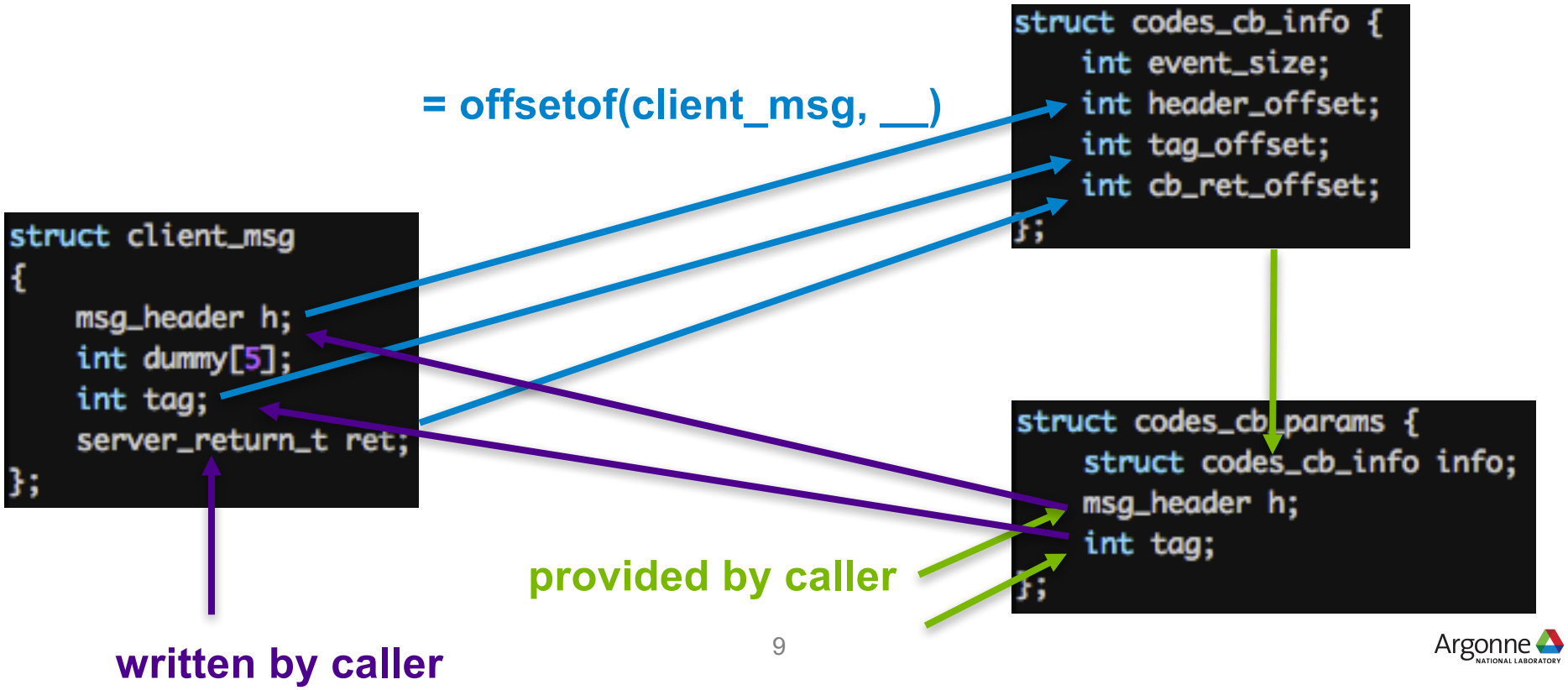
```
struct client_msg
{
    msg_header h;
    int dummy[5];
    int tag;
    server_return_t ret;
};
```

Full example at soc talk repository, callback.c

CALLBACK API

Convention for RPC-style event management

- Helpers (codes/codes-callback.h)
  - Offsets for header, tag, return value in callee
  - Callee parameter set including offsets, tag, header
  - macros to setup structures



MAPPING CONTEXT API

Managing implicit LP messaging

- Problem: lack of flexibility in implicit LP mappings
  - modelnet, local storage model, resource LP

```
LPGROUPS
{
   MODELNET_GRP
   {
      repetitions="36";
      server="10";
      modelnet_dragonfly="2";
      modelnet_dragonfly_router="1";
   }
}
```

Default Mapping

Server	Dragonfly
0	0
1	1
2	0
3	1
...	...

What about...?

Server	Dragonfly
0	0
1	0
...	...
4	0
5	1

Full example at soc repo, mapctx.c

MAPPING CONTEXT API

Managing implicit LP messaging

- Solution: provide a small set of deterministic contexts in which implicit LP mapping is performed
  - (codes/codes-mapping-context.h)
  - Modelnet, local storage model accepts mapping contexts
- Options:
  - Default: modular ID arithmetic
  - Ratio-apportioned mapping (5:1 in example)
  - Direct group-relative ID
  - Escape hatch: direct LP-ID mapping

Default mapping

Server	Dragonfly
0	0
1	1
2	0
3	1
...	...

```
LPGROUPS
{
   MODELNET_GRP
   {
      repetitions="36";
      server="10";
      modelnet_dragonfly="2";
      modelnet_dragonfly_router="1";
   }
}
```

Ratio-apportioned mapping

Server	Dragonfly
0	0
1	0
...	...
4	0
5	1

MULTI-JOB WORKLOADS

Full example at soc repo, jobmap.c

- Problem: CODES workload API only worked for a single application at a time
- Solution: job mapping API - map from job/rank to flat ID space (representing cores, nodes)
  - MPI replay simulation updated to support concurrent workloads
  - scripts provided for allocating system resources to applications

```
// trivial jobmap (can used as a simple default)
// single job, 5 ranks
struct codes_jobmap_params_identity identity_params = { 5 };
struct codes_jobmap_ctx *identity =
    codes_jobmap_configure(CODES_JOBMAP_IDENTITY, &identity_params);
printf("# IDENTITY JOBMAP\n");
print_jobmap(identity);

// jobmap that reads from file
struct codes_jobmap_params_list list_params = { "conf/jobmap.conf" };
struct codes_jobmap_ctx *list =
    codes_jobmap_configure(CODES_JOBMAP_LIST, &list_params);
printf("# LIST JOBMAP\n");
print_jobmap(list);
```

"Identity" mapping
- rank i – node i

"List" mapping
- Line per job
- i-th entry – node for rank i

```
0 1 2
5 3 4
7 9 11 13  6 8 10 12
```

MODELNET SAMPLING API

Point-in-time view of system state

- Problem: end-of-sim network aggregates are useful, but miss the full picture. Need finer-grained system (LP) characterization
- Solution: periodically sample state of modelnet LPs
  - simple as adding call “model_net_enable_sampling(interval, end_time)” before simulation starts
- Binary format, so can’t show it here ☺
- Currently only supported in the dragonfly model

SUMMARY

SUMMARY

- Model improvements
  - dragonfly, torus fidelity
  - data gathering
- Model composition, workload improvements
  - mapping improvements
  - event flow conventions
  - multi-app workloads

- Feedback welcome!


THANKS!

EXTRAS

Full example at soc talk repository, callback.c

CALLBACK API

Convention for RPC-style event management

- Example macro – GET_INIT_CB_PTRS
  - m->cb – codes_cb_params passed to callee
  - mret – event to write output to
  - lp->gid - LP ID of the callee
  - h, tag, rc – pointer declarations for header, tag, return variable
  - server_return_t - return type

```
struct codes_cb_info {
    int event_size;
    int header_offset;
    int tag_offset;
    int cb_ret_offset;
};
```

```
struct codes_cb_params {
    struct codes_cb_info info;
    msg_header h;
    int tag;
};
```

```
static void server_event(
        server_state_t *s,
        tw_bf *b,
        server_msg_t *m,
        tw_lp *lp)
{
    // assertions to check validity of callback structure relative to ret
    // type
    SANITY_CHECK_CB(&m->cb.info, server_return_t);

    tw_event *e = tw_event_new(m->h.src, 1., lp);
    void *mret = tw_event_data(e);

    // set callback and return
    GET_INIT_CB_PTRS(&m->cb, mret, lp->gid, h, tag, rc, server_return_t);
    *rc = (*s)++;

    tw_event_send(e);
}
```