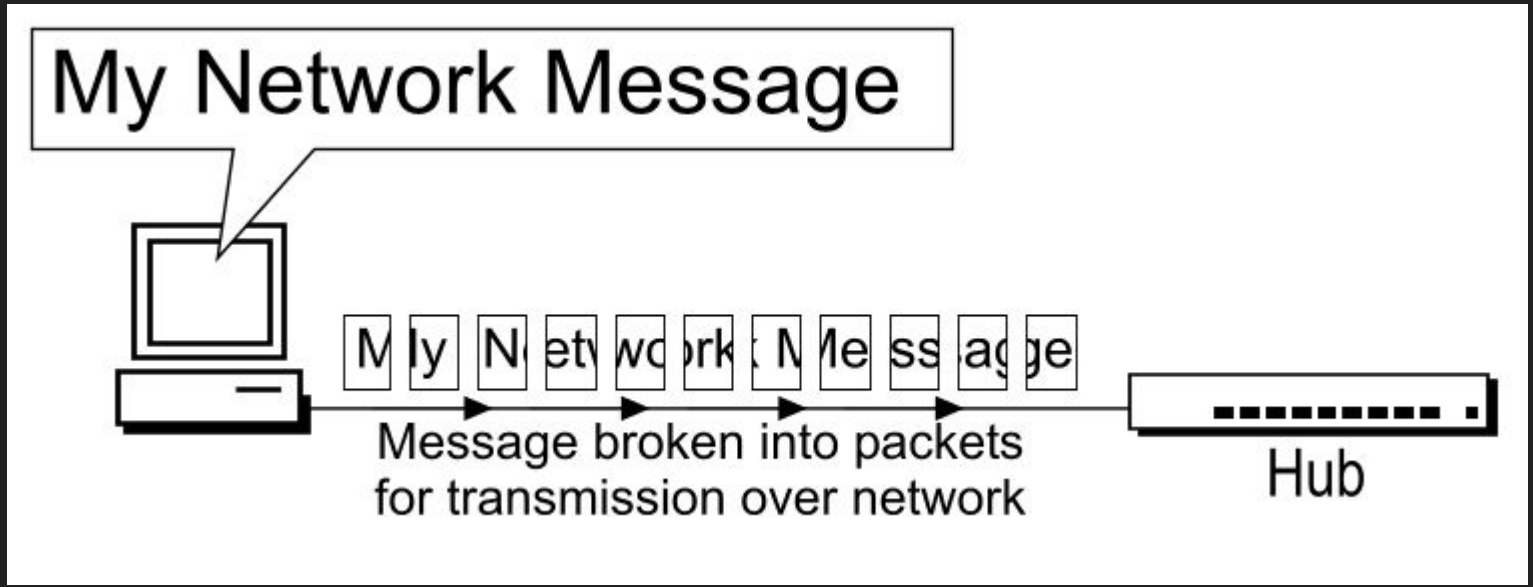
Homa

Data Center Transport Protocol

Presentation 3

Harsh Kapadia

Recap: Message vs Packet



Recap: Homa Features

- Message-oriented (RPCs)
- Connectionless
- Shortest Remaining Processing Time (SRPT) Scheduling
- Receiver-driven Congestion Control
- High out-of-order packet tolerance
- No per-packet acknowledgements
- At-least-once semantics

Recap: Sender vs Receiver

- Client → Server
  - Sender: Client
  - Receiver: Server
- Server → Receiver
  - Sender: Server
  - Receiver: Client

Recap: Homa Protocol Overview

- On receiving message from top layer, sender blindly sends unscheduled portion
- Sender can send further scheduled `DATA` packets only if receiver authorises through `GRANT` packet
- `GRANT` usually requests for '`RTT` bytes' worth outstanding data to keep transmission uninterrupted

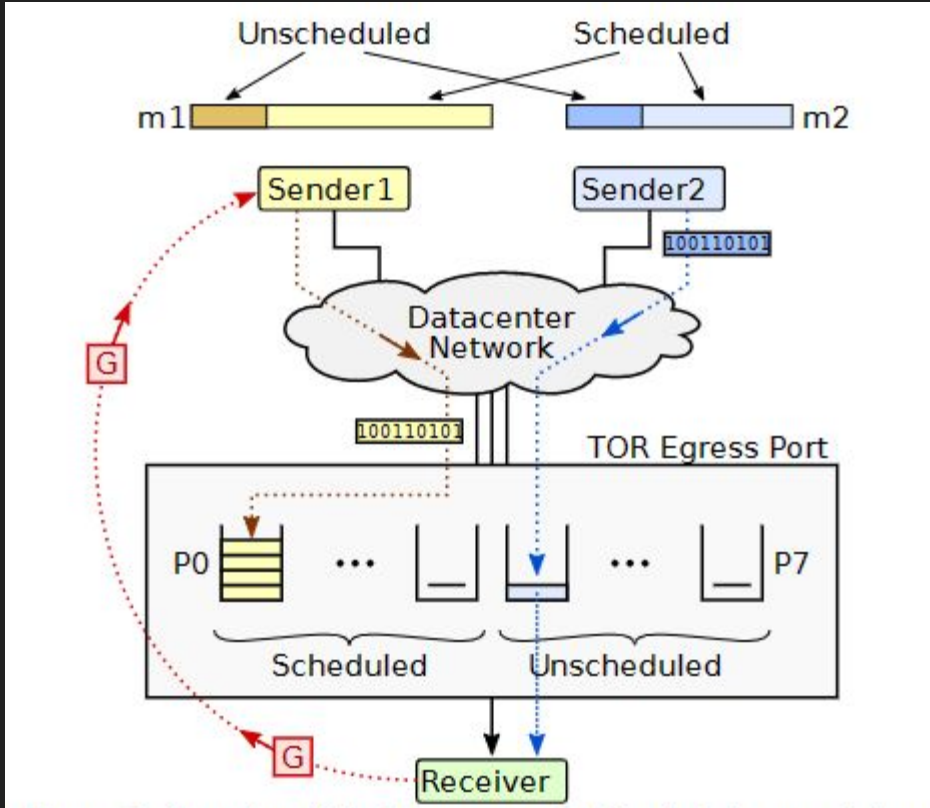
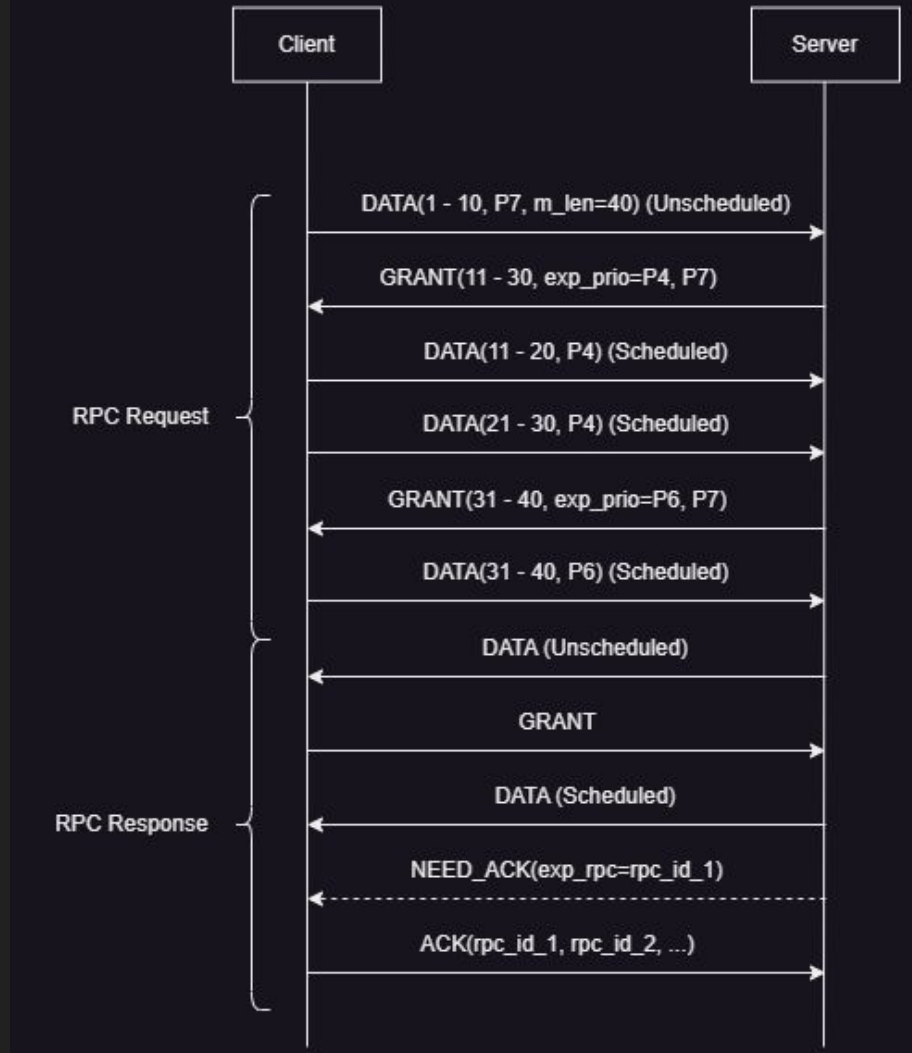


Figure 2: Overview of the Homa protocol. Sender1 is transmitting scheduled packets of message *m1*, while Sender2 is transmitting unscheduled packets of *m2*.

Recap: Homa Working

- RPC Request, RPC Response
- `DATA`, `GRANT`, `ACK`, `NEED_ACK`
- Priority levels: `P0` (lowest) to `P7` (highest)



Recap: Homa Linux Architecture

- Transmit (top): `homa_send()` → copy packets → TSO/GSO → Homa, IP layer → NIC
- Receive (bottom): NIC (RSS) → Interrupt → NAPI (GRO, SoftIRQ core choosing) → SoftIRQ (network stack traversal) → copy packets → `homa_recv()`

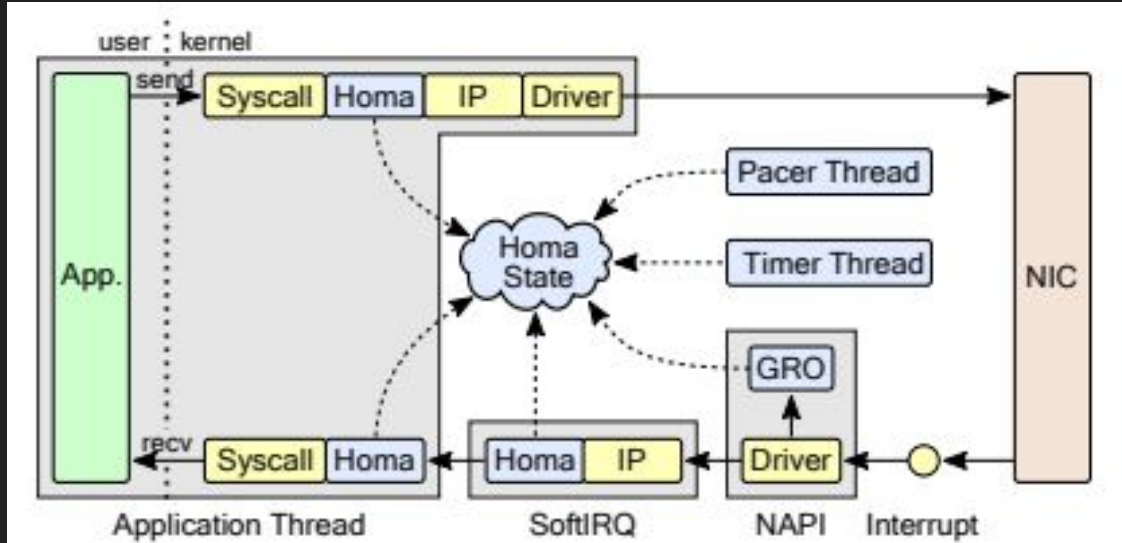


Figure 2: Structure of Homa/Linux. Homa components are shown in blue; existing Linux kernel modules are in yellow. Gray areas represent different cores. Only the primary sending and receiving paths are shown; other Homa elements such as the pacer thread and timer thread also transmit packets.

Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Sorting RPCs and Peers

- `homa_state` ⟶ `grantable_peers`
- `peer` ⟶ `grantable_rpcs`
- Sorting precedence for RPCs and peers
  a. `bytes_remaning`
  b. `birth_time`
- `bytes_remaining` ↓ ⇒ `priority` ↑
- `birth_time` ↓ (older) ⇒ `priority` ↑ (Tie-breaker mechanism)
- Sorting order
  a. RPCs
  b. Peers

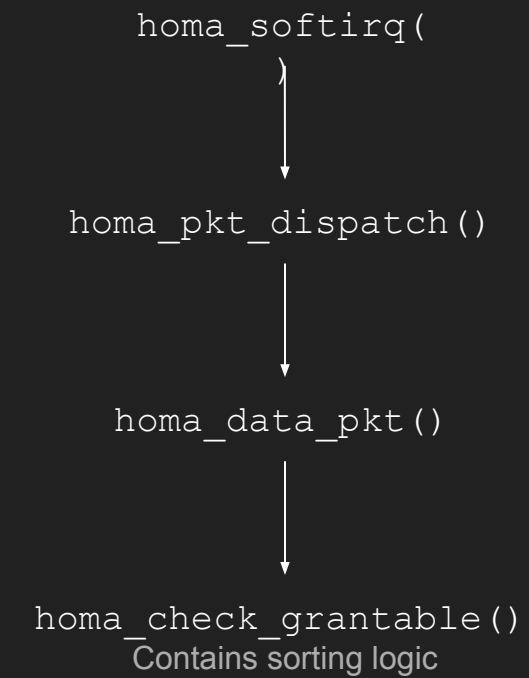
position_rpc.c

```
position_rpc(homa_state, rpc_to_check) {
    peer_to_check = rpc_to_check->peer;

    while(rpc in peer_to_check->grantable_rpcs) {
        if(rpc->bytes_remaining > rpc_to_check->bytes_remaining) {
            // Add `rpc_to_check` before `rpc`.

            position_peer(homa_state, peer_to_check);
            break;
        }
        else if(rpc->bytes_remaining == rpc_to_check->bytes_remaining) {
            if(rpc->birth > rpc_to_check->birth) {
                // Add `rpc_to_check` before `rpc`.

                position_peer(homa_state, peer_to_check);
                break;
            }
        }
    }
}
```

position_peer.c

```
position_peer(homa_state, peer_to_check) {
    first_rpc_in_peer_to_check = get_list_head(peer_to_check);

    while(peer in homa_state->grantable_peers) {
        first_rpc_in_peer = get_list_head(peer);

        if(first_rpc_in_peer->bytes_remaining > first_rpc_in_peer_to_check->bytes_remaining) {
            // Add `peer_to_check` before `peer`.

            break;
        }
        else if(first_rpc_in_peer->bytes_remaining ==
                                                first_rpc_in_peer_to_check->bytes_remaining) {
            if(first_rpc_in_peer->birth > first_rpc_in_peer_to_check->birth) {
                // Add `peer_to_check` before `peer`.

                break;
            }
        }
    }
}
```

RPC and Peer Sorting Timing

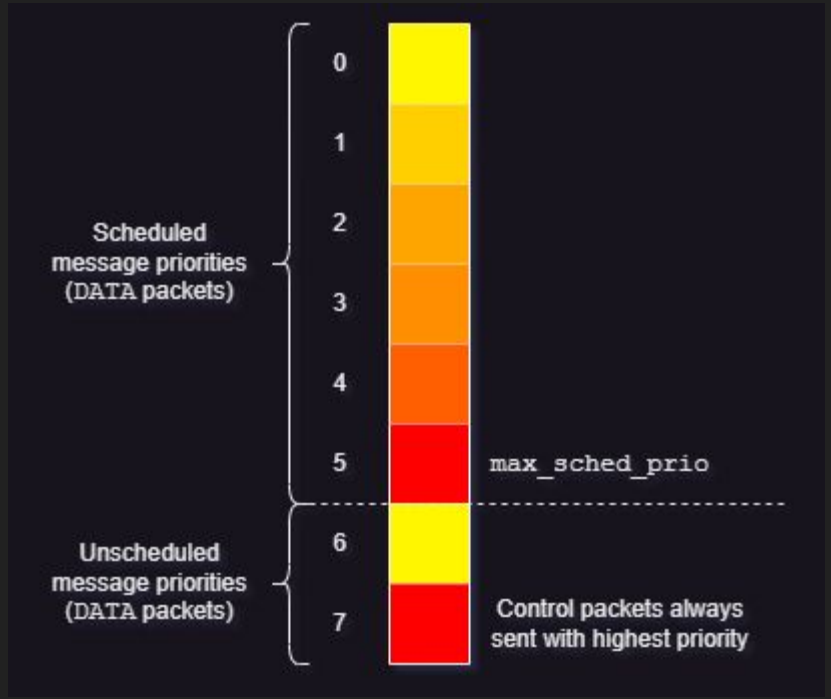


Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Calculating Scheduled Priorities

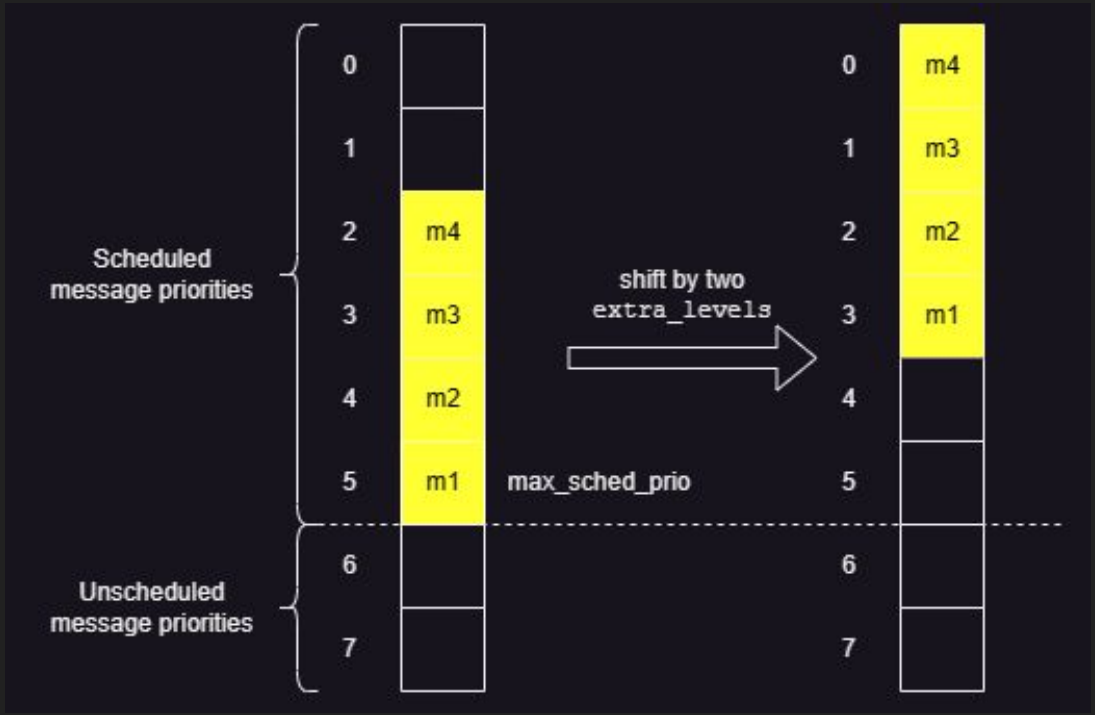
- Eight default priority levels: 0 (lowest) to 7 (highest)
- `max_sched_prio` splits levels between scheduled and unscheduled message (`DATA` packet) priorities
- Message length ↓ ⇒ Priority ↑
- Example:
  - Levels: 0 to 7
  - `max_sched_prio` = 5
  - ⇒ Sched prio levels = 0 to 5 (both inclusive)
  - ⇒ Unsched prio levels = 6 and 7
- To accommodate higher priority messages for SRPT, Homa assigns lowest possible priority to each RPC
- Calculated individually for first RPC of every grantable peer
  - Only first RPC of every grantable peer is granted



set_scheduled_data_priority.c

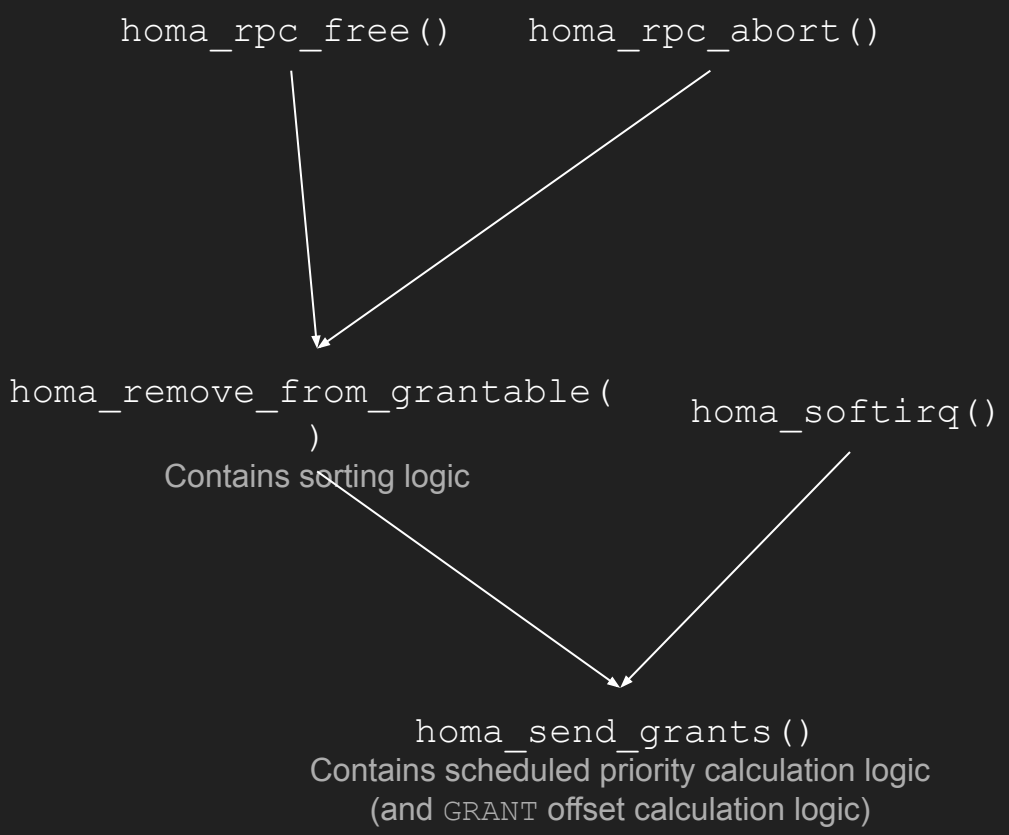
```
set_scheduled_data_priority(homa_state) {
    rank = 0;
    max_grants = 10;
    max_scheduled_priority = homa_state->max_sched_prio;
    num_grantable_peers = homa_state->num_grantable_peers;

    while(peer in homa_state->grantable_peers) {
        rank++;

        priority = max_scheduled_priority - (rank - 1);

        total_levels = max_scheduled_priority + 1;
        extra_levels = total_levels - num_grantable_peers;

        if (extra_levels >= 0) {
            priority = priority - extra_levels;
        }
        if (priority < 0) {
            priority = 0;
        }

        // Assign `priority` to `GRANT` packet.

        if(rank == max_grants) {
            break;
        }
    }
}
```

Lowest Priority Assignment



Scheduled Priority Calculation Timing

- Alt. title: ‘`GRANT` Packet Transmission Timing’
- `max_sched_prio` set by `unsched_cutoffs` array
  - More info. in slides ahead
- Same function call tree for GRANT offset calculation
  - More info. in slides ahead

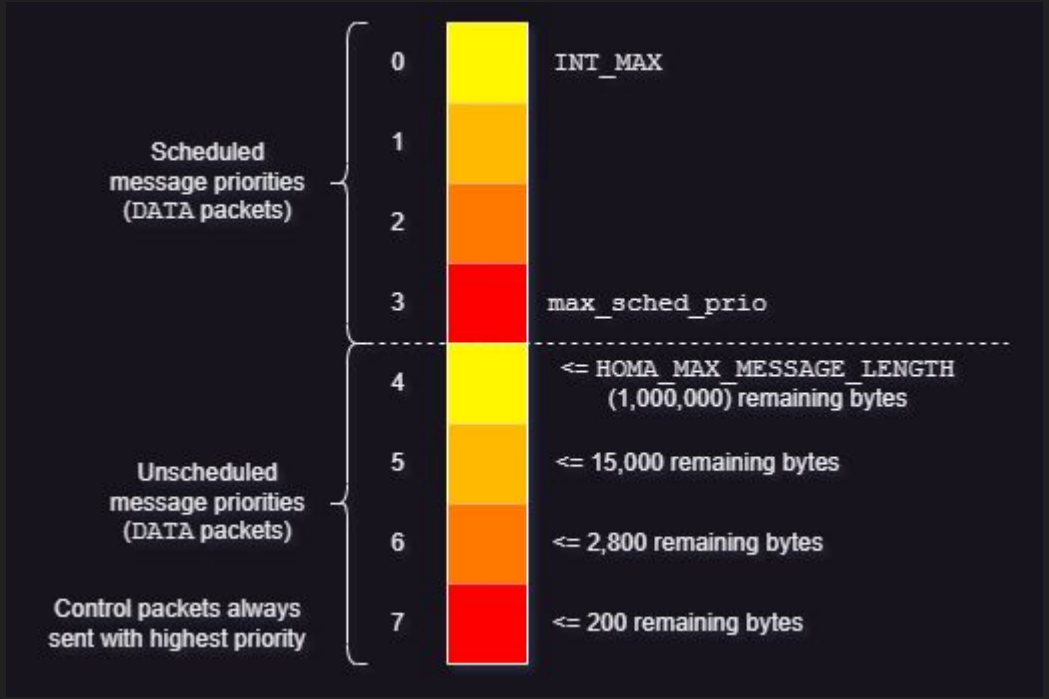


Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Calculating Unscheduled Priorities

- Decided by `unsched_cutoffs` array
- Statically defined values for unscheduled messages (`DATA` packets)
  - Control packets always sent at highest priority level
- `max_sched_prio = HOMA_MAX_MESSAGE_LENGTH - 1`



`unsched_cutoffs` array initialization
`cutoff_version` = 1

Cutoffs Initialization

Receiver: Incoming `DATA` packet `cutoff_version` is checked with `homa_state->cutoff_version`

homa_state.c

```
homa_state->num_priorities = HOMA_MAX_PRIORITIES; // = 8

homa_state->max_sched_prio = HOMA_MAX_PRIORITIES - 5; // = 3 (0 - 3 | 4 - 7)

homa_state->unsched_cutoffs[HOMA_MAX_PRIORITIES-1] = 200; // Level 7
homa_state->unsched_cutoffs[HOMA_MAX_PRIORITIES-2] = 2800; // Level 6
homa_state->unsched_cutoffs[HOMA_MAX_PRIORITIES-3] = 15000; // Level 5
homa_state->unsched_cutoffs[HOMA_MAX_PRIORITIES-4] = HOMA_MAX_MESSAGE_LENGTH;
                                                                // Level 4
homa_state->cutoff_version = 1;
```

Sender: Outgoing `DATA` packet `cutoff_version` is added from `peer->cutoff_version`

(Every receiver will have a different priority, so each `peer` structure on sender has to track its receiver data.)

peer.c

```
peer->unsched_cutoffs[HOMA_MAX_PRIORITIES-1] = 0; // Level 7
peer->unsched_cutoffs[HOMA_MAX_PRIORITIES-2] = INT_MAX; // Level 6

peer->cutoff_version = 0; // No `CUTOFFS` packet ever received

peer->last_update_jiffies = 0;
```

Sending `DATA` Packets

- Initialized `cutoff_version = 0` will cause a mismatch with receiver, which is initialized with `cutoff_version = 1`.
- Version mismatch causes receiver to send `CUTOFFS` packet.
  - More info. on next slide.

sender.c

```
if(DATA packet) {
    if(rpc->bytes_to_be_sent_offset < rpc->unscheduled_bytes_offset) {
        priority = get_unscheduled_priority(rpc->total_msg_length);
    }
    else {
        priority = rpc->scheduled_priority;
    }
}
else {
    priority = homa_state->total_num_of_priorities - 1;
}
```

Receiving `DATA` Packets

- Receiver will always have latest `cutoff_version` as it is the driver of communication and it updates priorities.
- Time (unit: Jiffies) is checked to not send multiple `CUTOFFS` packets to same sender when packets are continuously arriving

receiver.c

```
if(incoming_pkt_data_header->cutoff_version != homa_state->cutoff_version) {
    // Implies that the sender has a stale `CUTOFFS` packet.

    if(time_now != peer->last_cutoffs_packet_send_time) {
        /* Send updated `homa_state->unsched_cutoffs` array and
         * `homa_state->cutoff_version` in a `CUTOFFS` packet to peer.
         */

        // Update time (jiffies) when last `CUTOFFS` packet was sent.
    }
}
```

Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Calculating `rtt_bytes`

- Helps in keeping link utilization ~100%
- Unfortunately, currently a static config. parameter in receiver for all its peers
- Should ideally be frequently dynamically calculated per peer by receiver

Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Calculating `GRANT` Offset

- Offset indicates permission to send bytes up to a particular bytes
- `offset ≤ total_msg_length`
- Optimal amount of newly granted data: `rtt_bytes`
  - Helps to keep link utilization ~100%
- Example (At sender)
  - Total message length = 100
  - Sent data offset = 20
  - **`GRANT` offset received** = 45
  - ⇒ Data sent = bytes 21 to 45 (should be ~`rtt_bytes`)

set_data_offset.c

```
set_data_offset(homa_state) {
    counter = 0;
    max_grants = 10;
    total_bytes_available_to_grant = homa_state->max_incoming - homa_state->total_incoming;

    if(total_bytes_available_to_grant <= 0) {
        return;
    }

    while(peer in homa_state->grantable_peers) {
        counter++;

        first_rpc_in_peer = get_list_head(peer);
        rpc_bytes_received = first_rpc_in_peer->total_msg_length -
                                                    first_rpc_in_peer->msg_bytes_remaining;
        offset = rpc_bytes_received + homa_state->rtt_bytes;

        if(offset > first_rpc_in_peer->total_msg_length) {
            offset = first_rpc_in_peer->total_msg_length;
        }

        increment = offset - first_rpc_in_peer->total_incoming_bytes;
```

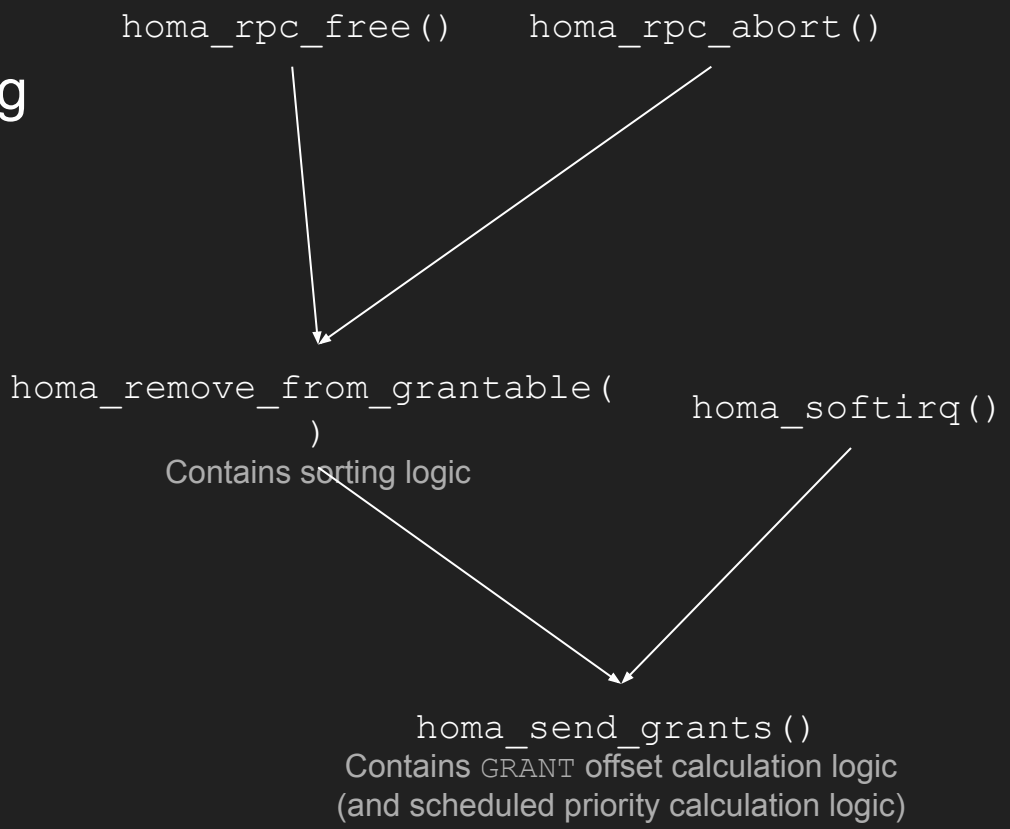
continued…

…continued

```
        if (increment <= 0) {
            continue;
        }
        if (total_bytes_available_to_grant <= 0) {
            break;
        }
        if (increment > total_bytes_available_to_grant) {
            increment = total_bytes_available_to_grant;
            offset = first_rpc_in_peer->total_incoming_bytes + increment;
        }

        first_rpc_in_peer->total_incoming_bytes = offset;
        total_bytes_available_to_grant = total_bytes_available_to_grant - increment;

        // Assign `offset` to `GRANT` packet.

        if(counter == max_grants) {
            break;
        }
    }
}
```

`GRANT` Offset Calculation Timing

- Alt. title: '`GRANT` Packet Transmission Timing'
- `max_sched_prio` set by `unsched_cutoffs` array
  - More info. in slides ahead
- Same function call tree for Scheduled Priority Calculation
  - As already seen in slides before



Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

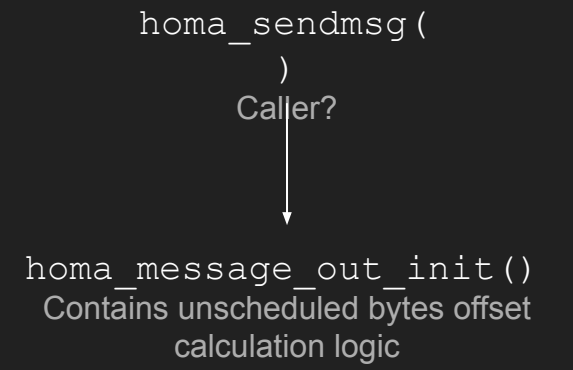
Calculating Unscheduled Bytes Offset

- Calculated at sender
- Example (At sender)
  - Total message length = 100
  - Calculated unschedules bytes offset = 20
  - ⇒ Data sent = bytes 1 to 20 (should at least be ~`rtt_bytes`)
- Rationale behind formula to decide unscheduled bytes offset not clear

set_unscheduled_byte_offset.c

```
set_unscheduled_byte_offset(rpc) {
    mss = mtu - ip_header_length - data_header_length;

    if(rpc->total_msg_length <= mss) {
        rpc->unscheduled_bytes = rpc->total_msg_length;
    }
    else {
        gso_pkt_data = pkts_per_gso * mss;

        rpc->unscheduled_bytes = rpc->rtt_bytes + gso_pkt_data - 1;

        extra_bytes = rpc->unscheduled_bytes % gso_pkt_data;
        rpc->unscheduled_bytes = rpc->unscheduled_bytes - extra_bytes;

        if (rpc->unscheduled_bytes > rpc->total_msg_length) {
            rpc->unscheduled_bytes = rpc->total_msg_length;
        }
    }
}
```

Unscheduled Bytes Offset Calculation Timing



Recap: Homa Algorithms

- Sorting RPCs and Peers
- Calculating Scheduled Priorities
- Calculating Unscheduled Priorities
- Calculating `rtt_bytes`
- Calculating `GRANT` Offset
- Calculating Unscheduled Bytes Offset

Things That Did Not Work Out

- Building Homa module
  - Multiple errors that could not be debugged
  - Will have to ask Prof. Ousterhout for help
- Setting up CloudLab environment
  - Sign in successful
  - Allocating exact infrastructure with Switch control tricky
  - HomaModule GitHub repo does not have sufficient instructions to set up environment

Tentative Future Plans

- Ask Prof. Ousterhout for help
- Set up CloudLab environment
- Build Homa module
- Conduct Homa experiments

Resources

- Homa: networking.harshkapadia.me/homa
- Code image generator: ray.so
- Array diagrams: draw.io

Thank you!