The program begins in the load balancer, which spawns a set of worker processes and binds channels to them. Each worker has two channel connections to the load balancer: a packet channel for work requests and a stream channel for acks. When work arrives on the packet channel from the load balancer, the worker processes it and then sends back an ack to the load balancer. The load balancer will not send new work to a worker unless an 'ack' word is available.

The worker's packet processing algorithm uses MRAPI to accomplish the following tasks:

1. At the start of time, it allocates a shared memory hash table that contains linked lists of packet flows.

2. As each packet arrives, the worker computes a hash bucket number based on its source and destination addresses.

3. The worker locks that hash bucket.

4. Having gained exclusive access to that bucket, the worker scans a linked list to see if the flow already exists, and if so, updates it.

5. If the flow is new to the system, the worker allocates a flow info object from MRAPI shared memory and adds it to the list.

6. The worker releases the lock on the hash bucket.

For ease of reference, the relevant MRAPI worker code is shown below. This code is run on several 'worker' cores, all accessing the same shared memory flow table in parallel. The code below includes both the shared memory initialization code run on all workers and the function that each worker calls when a packet arrives.

```
#define MY_SHMEM_ID 7

#define MAX_FLOWS (1024 * 1024)

#define HASH_BUCKETS 512


typedef struct flow_info_s {

  flow_info_s *next;

  ... various protocol state ...

} flow_info_t;


typedef struct {

  mrapi_mutex_hndl_t lock;

  flow_info_t *list;

} flow_hash_table_entry_t;
```