```cpp
switch::handle_payload(event* ev){
  packet* pkt = cast(ev);
  router_->route(pkt);
  xbar_->handle_payload(pkt);
}
```

```cpp
minimal_router::
route(packet* pkt)
{
    pkt.vc  = comp vc;
    pkt.outport = comp port;
}
```

```cpp
xbar::handle_payload(packet* pkt){
  pkt->set_arrival(now);
  int vc = pkt->next_vc();
  int port = pkt->next_port();
  //check if we have enough credits
  int& avail_credits = credits(vc, port);
  if (avail_credits >= pkt->num_bytes()){
    avail_credits -= pkt->num_bytes();
    input& in = inputs_[pkt->inport()];
    output& out = outputs_[pkt->outport()];
    send(arb, pkt, in, out);
  } else {
    queue packet for when credits arrive
  }
}
```

```cpp
struct packet_arbitration_st {
    //the packet being arbitrated
    packet* pkt;
    //the time arbitration occurs
    timestamp now;
    //time first flit of packet leaves
    timestamp head_leaves;
    //time last flit of packet leaves
    timestamp tail_leaves;
    //time credit leaves for source
    timestamp credit_leaves;
    int src_outport; //ports to use
    int dst_inport;
};
```

```cpp
sender::send(arbitrator* arb,
    packet* pkt, input src, output dst){
  //setup struct that holds p
  packet_arbitration_st st;
  configure the packet stats struct
  arb->arbitrate(st); //pass by reference
  stat_collector_->collect_single_event(st);
  //send a credit back to src
  send_credit(src, pkt st.credit_leaves);
  //if this is the tail packet in a flow
  //might need flow-level ack
  if (pkt->is_tail()){
    schedule(st.tail_leaves, ack_handler_,
             pkt->clone_ack());
  }
  //when packet head will arrive at next switch
  timestamp arrival = st.head_leaves + send_lat_;
  schedule(arrival, dest.handler, pkt);
}
```

```cpp
X::collect_single_event(
    const pkt_arbitration_t& st)
{
  //stats collection
}
```