RTT measurement implementation using spin bit & co.

Mirja Kühlewind, Brian Trammell

6. Plenary meeting Aberdeen, June 12, 2018

mami

measurement and architecture for a middleboxed internet

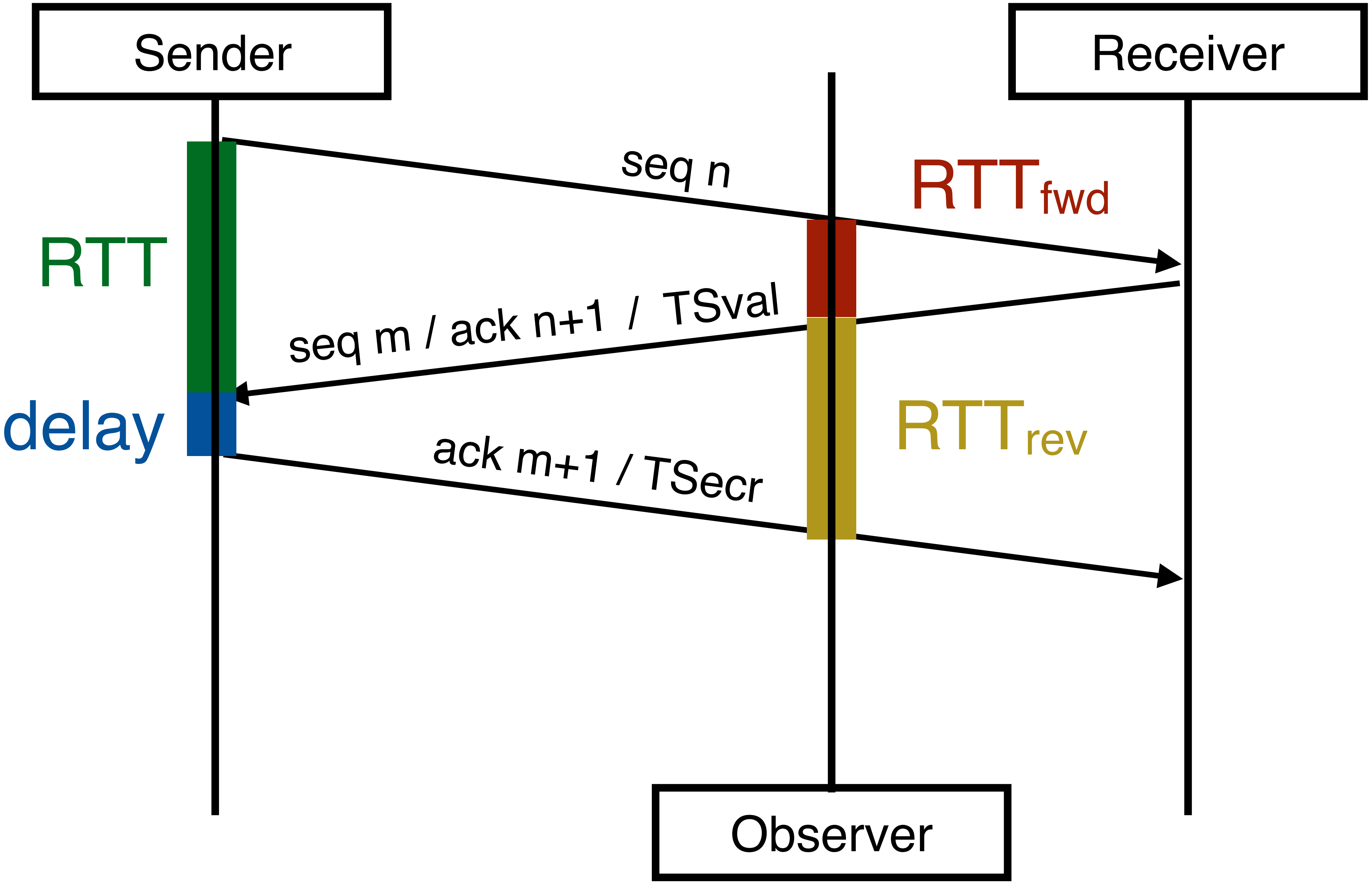
measurement architecture experimentation

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421.The opinions expressed and arguments employed reflect only the authors' view. The European Commission is not responsible for any use that may be made of that information.

RTT measurements

- TCP
  - SYN # and ACK # matching
  - TS option
  - Spin bit and VEC
    - using 3 remaining/reserved TCP header bits (no overhead)
    - New TCP option (3 bytes)
- QUIC
  - Spin bit and VEC
    - reserved bits in former short header type field (no overhead)
    - separate measurement byte (1 byte)
- PLUS
  - PN and PNE matching

RTT estimation with TCP



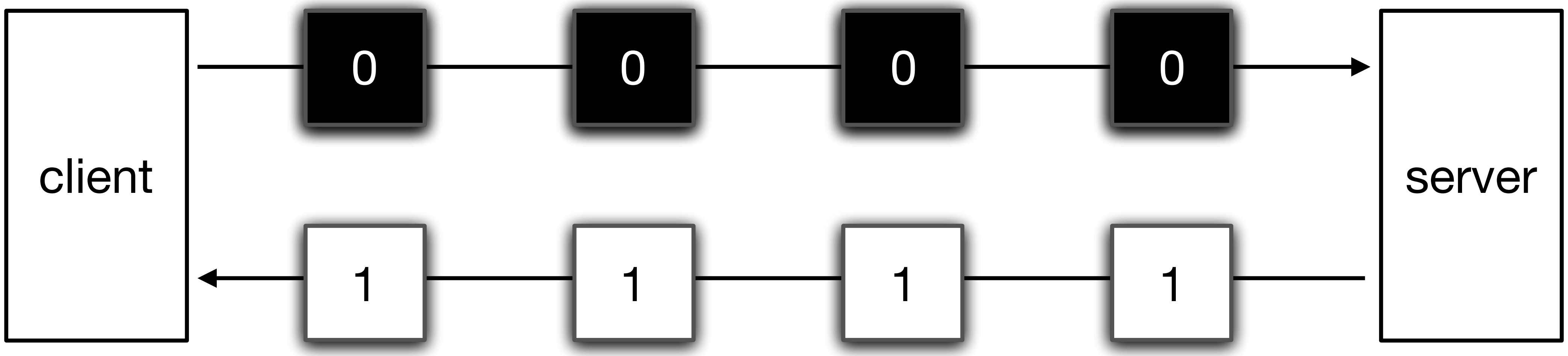
- Use of SEQ# and ACK# and/or TCP Timestamp Option

Replacing TCP RTT Measurement in the QUIC Wire Image: the Spin Bit

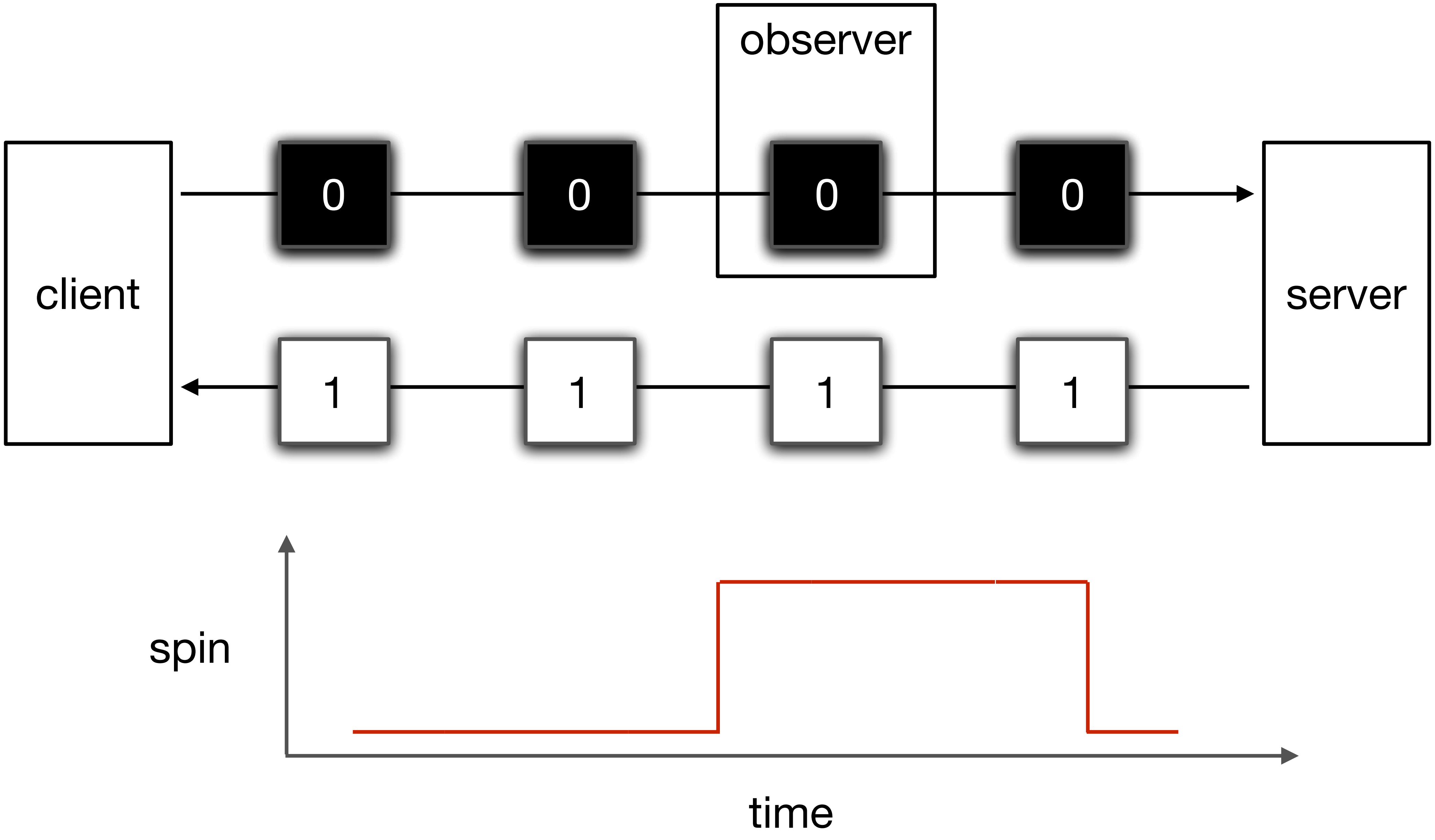
- Proposal: take a bit from QUIC short header type field and make it spin
- Server sets last spin it saw on each packet it sends
- Client sets ~(last spin it saw) on each packet it sends
- Creates a square-wave with period == RTT (when sender not app-limited)

```
+-+-+-+-+-+-+-+-+
|0|K|1|1|0|S V V|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Destination Connection ID (0..144)           ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Encrypted Packet Number (8/16/32)           ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Protected Payload (*)                   ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

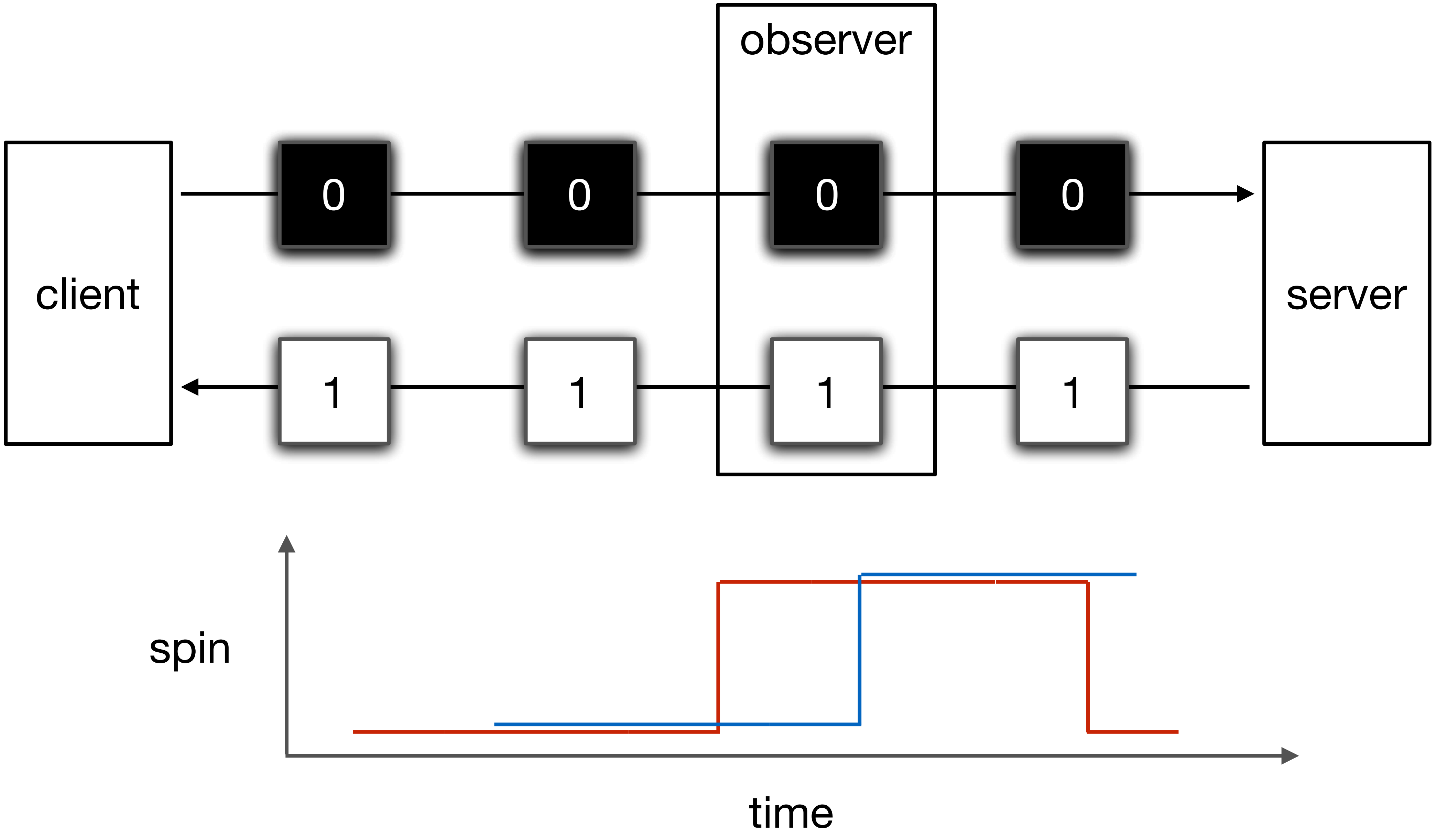
How does it work?



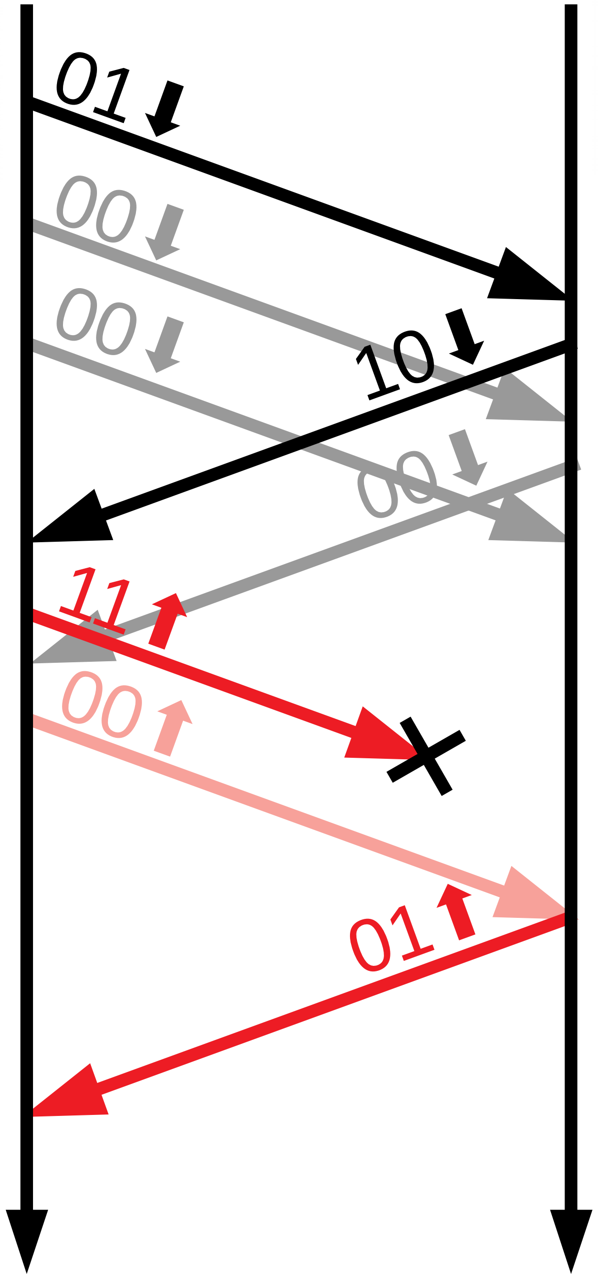
Unidirectional one-point measurement



Bidirectional one-point measurement



Dealing with Loss and Reordering: The Valid Edge Counter



- Bursty traffic can lead to wild overestimates of RTT: adds delay between bursts to actual measured RTT.
  - A damping filter can reduce overestimate samples
- Addition of a two-bit *valid edge counter* eliminates overestimation as well as fixing issues with packet loss and reordering:
  - On non-edge, delayed edge, edge on reordered packet: valid ← 00
  - On all other edges: valid ← last received valid + 1
  - Produces a 11 signal ("good edge") 1.5RTT after last reorder/delay, requires both sides to be reordering/delay-free, resets after an edge is lost.
- Rejects invalid samples due to bursty traffic, deals with reordering as well as two-bit spin, and adds tolerance to heavy burst losses, without PN visibility

The spin bit and Valid Edge Count (VEC)
draft-trammell-quic-spin & draft-ietf-quic-spin-exp

- **Spin bit**
  - Client/initiator spins by inverting the spin bit value that was received on the last packet from the server
  - Server reflects the same spin bit value as received in the last packet from the client
  - This generates a signal that has at most one “edge” (a transition 0 → 1 or 1 → 0) in flight
- **VEC**
  - By default, the VEC is set to 0.
  - If a packet contains an edge, and that edge is delayed (sent more than a configured delay since the edge was received, defaulting to 1ms), the VEC is set to 1.
  - If a packet contains an edge, and that edge is not delayed, the VEC is set to the value of the VEC that accompanied the last incoming spin bit transition plus one.
    - This counter holds at 3, instead of cycling around
    - If an edge received with a VEC of 0, it will be reflected as an edge with a VEC of 1; with a VEC of 1 as VEC of 2, and a VEC of 2 or 3 as a VEC of 3.
  - This mechanism allows observers to recognize spurious edges due to reordering and delayed edges due to loss, since these packets will have been sent with VEC 0.

Spin bit (and VEC) implementation

Update spin and VEC from incoming packet:

```
/* only considering in order packets */
if (PN >= PN_max) {
      /* edge detected */
      if (spin_next != spin_rcv) {
            vec_next = min(vec_rcv + 1, 3)
            t_last = t_sys
      }

       /* server reflects; client spins */
       if (is_initiator) {
            spin_next = !spin_rcv
      } else {
            spin_next = spin_rcv
      }

      PN_max = PN
}
```

Set spin and VEC on outgoing packet:

```
/* set spin to last observed spin value */
spin_snd = spin_next

/* reset VEC to 1 if last incoming packet
 * was observed more than delay_max ago */
if (t_sys - t_last > delay_max) {
        vec_snd = 1
} else {
        vec_snd = vec_next
}

vec_next = 0
```

Spin bit (and VEC) implementation in TCP

- New sysctl `net.ipv4.tcp.spin`
- Use SEQ# and ACK# instead of PN

- TODOs
  - VEC reset after delay_max not working properly
  - new sysctl from delay_max

- Next
  - Further improve re-order robustness…