# Keeping Time:

## Trust Domain Virtual Clock Distributor (TDVC)

Kathleen Nichols
nichols@pollere.net

## Context: Defined-trust Communications

- The Defined-trust framework employs concepts from and is inspired by a large body of previous work and grew out of work with NDN
- Defined-trust Domains are a type of Limited Domain (RFC8799) focused on security and using Defined-trust Transport (DeftT) protocol
- Rules specifying the networking of application information are defined in a communications schema that governs all information exchanged, including certificates of identity chains, and is integrated with the transport
- Member identities are distributed as a chain of trust, public certificates that have each been signed by the signing key associated with the next certificate in the chain and, at the root, signed by the trust anchor of the trust domain

Open source code base at https://github.com/pollere/DCT. Documentation at: https://pollere.net/publications.html, https://pollere.net/Txtdocs/defTrust.html, https://www.tdcommons.org/dpubs_series/7887

TDVC details are specific to Defined-trust Communications, but the exploitation of collection-based communication semantics to enable robust and efficient implementations can also apply to NDN

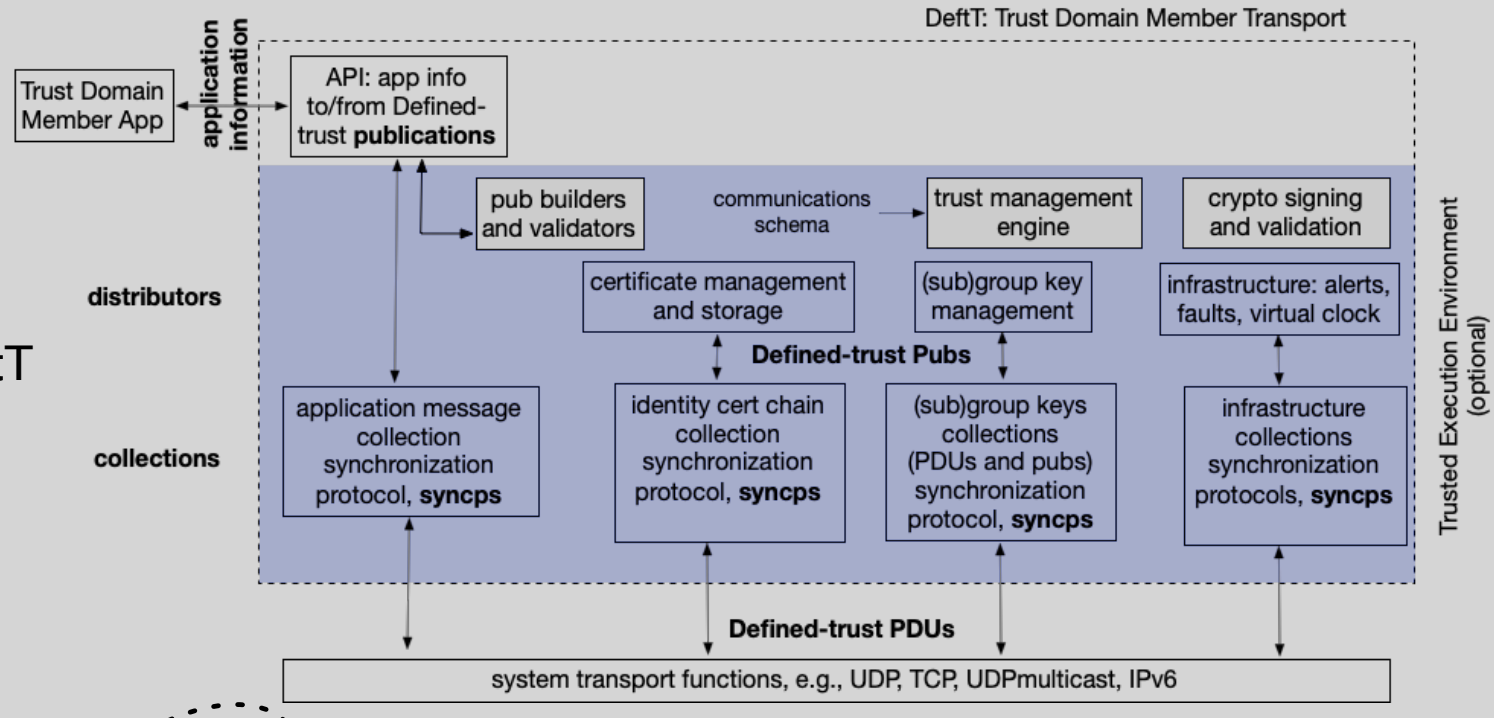
## Problem: Domain-wide clock synchronization

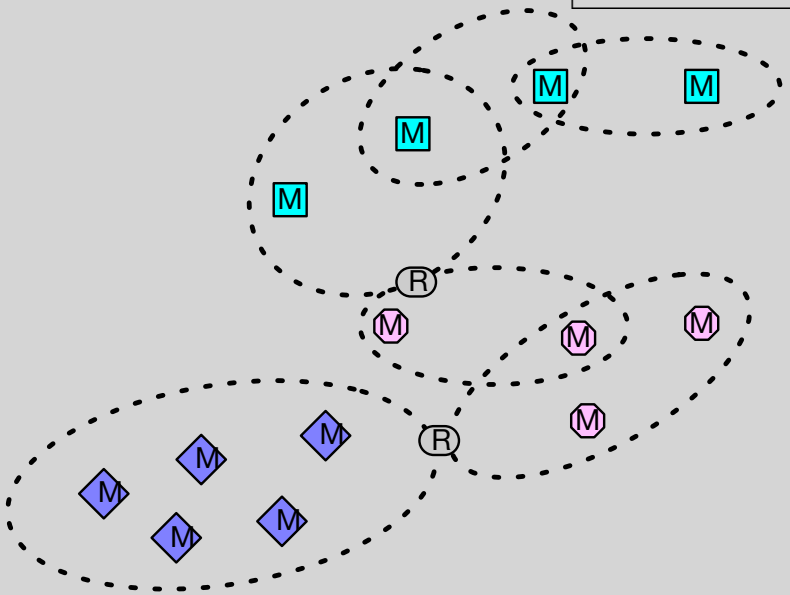
- DeftT provides a publish/subscribe API to collections of hierarchically named units of information (**publications**) (MQTT-like) along with its internal collections that are part of the transport's infrastructure
- Each collection is managed by its own instance of *syncps* that sends and receives publications wrapped in its own protocol data units (PDUs) which are exchanged using system transports, e.g., UDP, TCP, IPv6, LoRA
- PDUs are hierarchically named and prefixed with the domain identifier and their collection name
- Publications are **timestamped** to prevent replay attacks, among other uses, requiring a shared publication clock throughout the Trust Domain
  - » Scott Gray of Operant found many devices in the field do **not** use an NTP and do **not** use external communications

Solution: Give DeftT an integrated, self-contained virtual clock calibrated to its required accuracy

Note: DeftT publications and PDUs are both similar to NDN Data in format

# Diagram of DeftT




A Trust Domain may span subnets (color) interconnected by relays

Not all devices of a subnet may hear each other directly

## There are a lot of ways to calibrate clocks in a network

- Internet-based approaches (e.g., NTP) access globally accurate time servers
- Sensor network approaches designate or elect primary sources and build trees to distribute the reference time. Estimate skew and offset parameters relative to the reference.
- In our applications, neither approach is attractive; looking for something self-contained and robust to communication topology changes and failures. Further, we can approach calibration somewhat differently:
  - TDVC doesn't need to track "actual" time nor be monotonic over its history. Relative precision can be as large as ~0.5 seconds for the usual case of publications with a lifetime of 1 second (though aim for better)
  - using collection-based semantics, can ignore skew estimation and just periodically synchronize the **offset** of the virtual clock
  - the collection is updated with each member's virtual clock often enough to catch drifting out of tolerance (~10 minutes)

## Trust Domain Virtual Clock (TDVC) Overview

Certificate publications that make up identities have a lifetime in hours to years; thus, DeftT's initial membership **joining** process succeeds even when clocks differ

Timing problems can arise when exchanging publications with shorter lifetimes, e.g., setting up group encryption or sending application messages

TDVC is managed by a **distributor** module that starts after initial membership certificates have been exchanged (*joining*):

– uses a *collection* to provide a DeftT-specific clock integrated with transport
– does not need to be particularly close to actual time, relative precision of this clock can be as large as ~0.5 seconds for the usual case of publications with a lifetime of 1 second (but aim for better)
– algorithm is **completely distributed** between members
– **no** use of external servers
– members with access to precise clocks can be designated with a *capability* in the identity
– TDVC calibration is within single-hop reachable *neighborhoods* and propagates gossip-style (non-default syncps parameters)

## TDVC's calibration method - monitoring for tolerance

- Members periodically send their virtual clock as timestamps in publications. To keep the clock samples fresh and minimize delays, pubs:
  - can only be received by members reachable within a single-hop (a *neighbor*)
  - are not resent by others
  - are not valid long enough to be sent more than once
- Periodicity should be short enough for timely detection of out-of-tolerances but long enough to avoid flooding communication links
- Virtual clock publications contain the number of neighbor members (including the sending member itself) within tolerance of the sender's current virtual clock estimate and the state of the sender
- A newly started member sends its virtual clock (which will be identical to its system clock) with an *in-tolerance neighborhood size* (**nhSz**) of 1
- Members that receive a clock publication with a timestamp that differs from its local virtual clock by more than the tolerance enter a calibration cycle:
  - difference of received clock from local clock is saved, indexed by the unique identity of the sender
  - member sets its state to 2, sends a **set** of its own clock publications (~3 to 8)
  - incoming clock publications are processed (next slide)

## TDVC's calibration method - calibration cycle

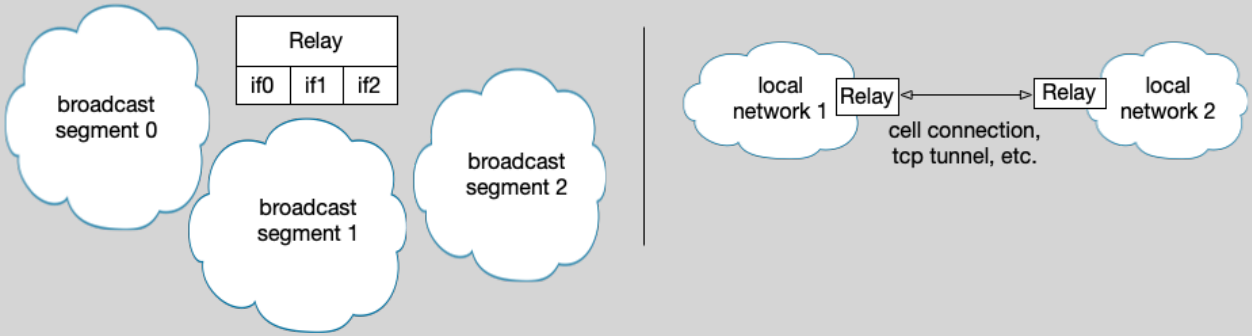
- Receiving members compute clock difference values (amount local virtual clock is ahead of sender, positive or negative)
  - if this is the first clock publication from this neighbor or if the neighbor's state has changed, save in a record at sender's id
  - else compare the difference to the stored value for the sender, replacing the old value if it is smaller
  - update the record with the sender's current **nhSz** and state
  - minimum received value is used to alleviate the non-determinism of processing and transport delays
- Members perform an offset computation a short delay after sending their set of clock publications
- Rounds of sending clock publication sets and computing offsets continue until all neighbors are within tolerance.
  - a member in-tolerance with all its neighbors moves to state 1 and must remain in tolerance through several rounds before moving to *calibrated* state 0
  - a member not in calibration increments its state (rolling over to 2)
- The extra delays are for networks that are not fully connected

# Offset computation

1. Replicate (in a vector) each neighbor's quantized minimum clock difference values by its **nhSz** less the minimum nhSz value plus 1. Local clock difference is represented by 0s.
2. If the absolute value of all clock differences are less than tolerance, set **state**=1 and **nhSz** is equal to the number of distinct neighbors heard from plus 1 (for self). Go to step 8.
3. Increment **state**, rolling over to 2 at 255.
4. Set **adjust** to the statistical *mode* of the clock differences vector. If more than one value at the mode, **adjust** = smallest non-zero absolute value difference. In case of a negative and positive at that value, use the negative, i.e., move in the forward clock direction. (other tie-breakers are possible)
5. If **adjust** is zero, check for an *impasse* where **adjust** has been zero and not all neighbors are in tolerance for a several rounds (e.g., ten). If an *impasse* and there are negative difference values, **adjust** = smallest absolute value
6. Set **nhSz** to the number of neighbors whose clock difference values are within tolerance of **adjust** plus one (for self since **adjust** will put the local virtual clock in tolerance) If **nhSz** is equal to the total size of the neighborhood (all members heard from), **state** = 1 (in tolerance and counting neighbors)
7. If **adjust** != 0, add **adjust** to the running virtual clock **calibrate** for this calibration cycle
8. The next set of clock publications use the updated **state** and **nhSz** and subtract **calibrate** from the local virtual clock to compute the publication timestamp.
9. When all neighbors have been in state 1 for a set number of rounds (~3), calibration round is complete, **state** = 0, local virtual clock -= **calibrate** and **calibrate** = 0

- Calibration restarts if an out-of-tolerance neighbor is detected (via periodic clock publications)
- Step 4 above can be implemented with approaches other than using the mode; e.g., a version using the median was also used but doesn't converge as well for non-fully connected networks

# Relays




Relays act as a single node or member in the virtual clock distribution. This is implemented by:

- Individual DeftTs of the relay pass all clock difference samples to the relay's application to store
- The calibration algorithm is run in the relay's application
- The final value of **offset** is passed to all the attached DeftTs

Relays may have subnets with significant transit delays, e.g., a cross-country TCP link, a LORA network

- non-negligible delays should be removed from the differences before passing to the relay application
- the TDVC contains its own RTT estimator for this purpose

## Processing and transit delay

Processing (cryptographic signing and validation) and transit delay add noise. TDVC addresses this in two ways:

- all clock differences are quantized, always rounding down. The quantization value chosen should be slightly greater than the normal transit/processing delay for broadcast networks
- uses a round-trip time (RTT) estimator for its neighborhood, subtracting this delay from clock differences when non-negligible (> quantization)

TDVC uses a neighborhood-based approach to RTT that works for both unicast and broadcast networks:

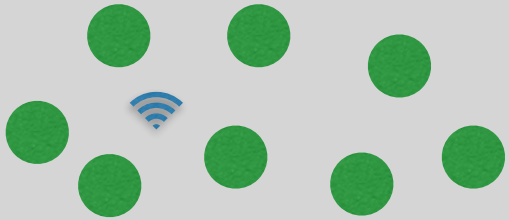
1. Members subscribe to both *ping* and *echo* publications.
2. At randomized periods, members send a *ping* publication.
   - selects a neighbor randomly (from the clock differences list)
   - id of that neighbor is used in the name of the *ping* publication
   - current time is saved, indexed by the unique id
3. *ping* receiving members check the id against their own:
   - if same, create an *echo* publication and send it
   - otherwise, save current time, indexed by id in the *ping*
4. *echo* receiving members obtain the senders's identity from the publication's signing chain, use it to locate the saved *ping* time, subtract it from the current time, and save to observed RTTs. (can be kept per-neighborhood or per-member)

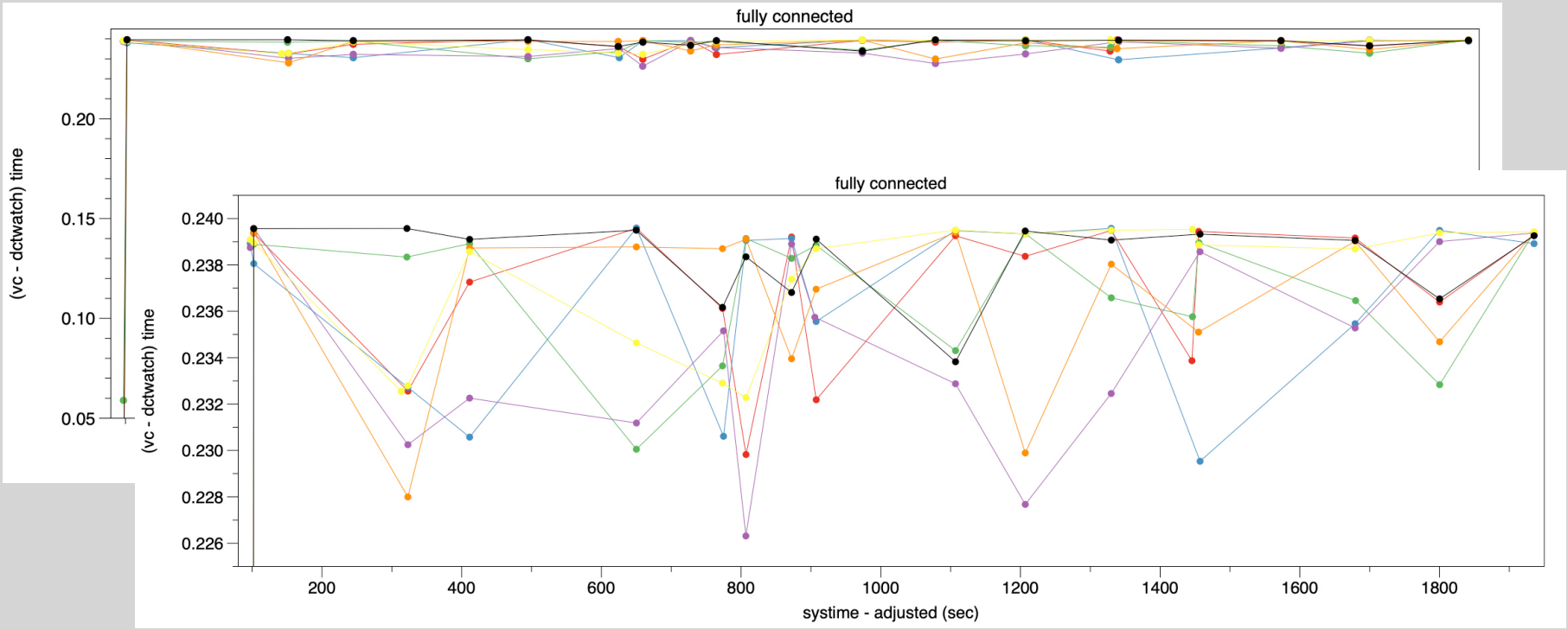
Neighborhood transit delay for broadcast networks can be used as a quantization value

## Notes on preliminary testing

- 8 members emulated on a single machine; rtts are < ~2msec
- uses 5ms quantization and 20ms difference tolerance
  - c*alibrate* to differences of +/-quantization
  - differences that *exceed tolerance* restart calibration
- each member gets a random offset at initialization random(-5,5)*60ms and a randomized drift emulation that adds (-2,2)*5ms at 2.5sec intervals (Clock drift is more typically expected to be ~40us/sec, 12 ms in 5 min)
- the large random periodic perturbations make the experiment's TDVC move more than expected in normal use
- DCT provides the ability to emulate three types of connectivity: full, two overlapping groups, and a linear arrangement (1 or 2 neighbors)
- member publishes a log message containing virtual clock when it finishes calibration. Subtracting this from the system clock (the dctwatch timestamp) charts members' differences (which should be in tolerance) from the system clock
- The first calibration convergences took +/-6sec for fully connected and 2groups, and +/-30sec for linear arrangement. Convergence delays include the built-in wait times

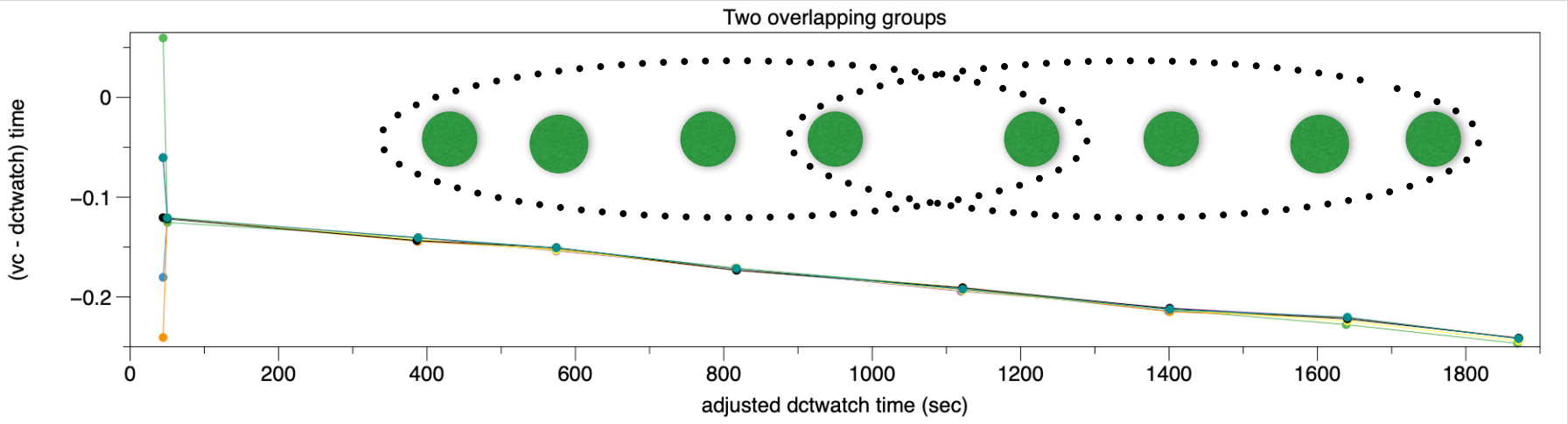
# Eight members fully connected via broadcast

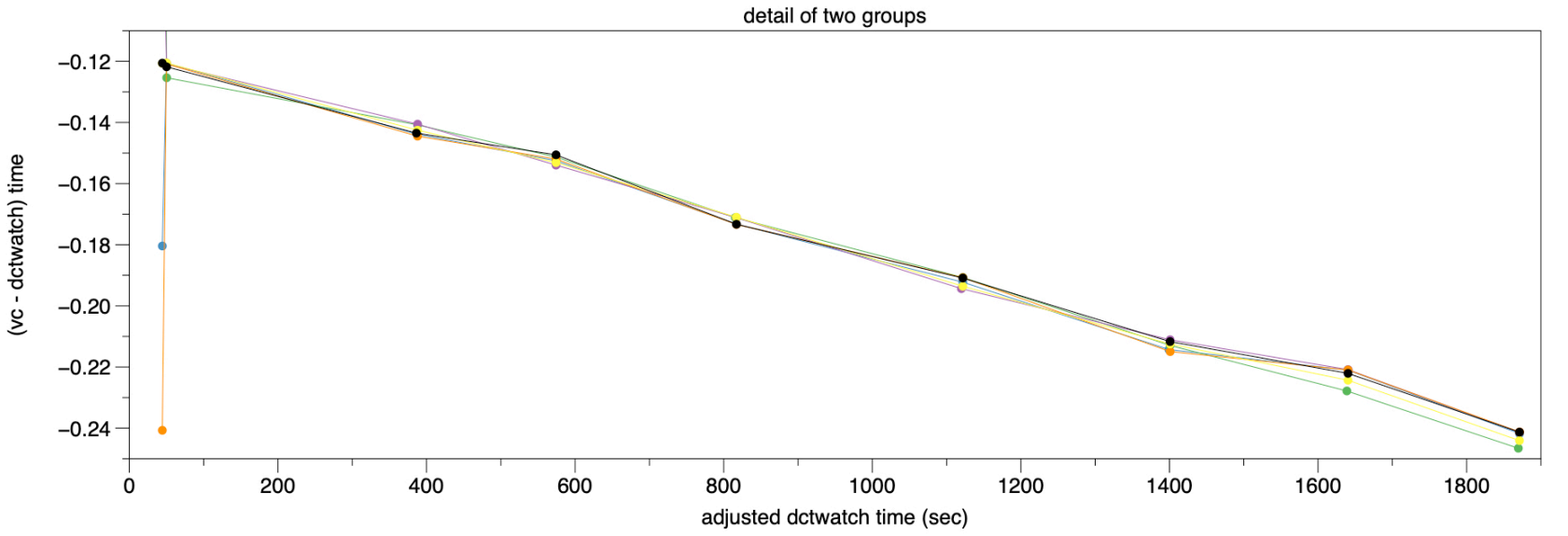




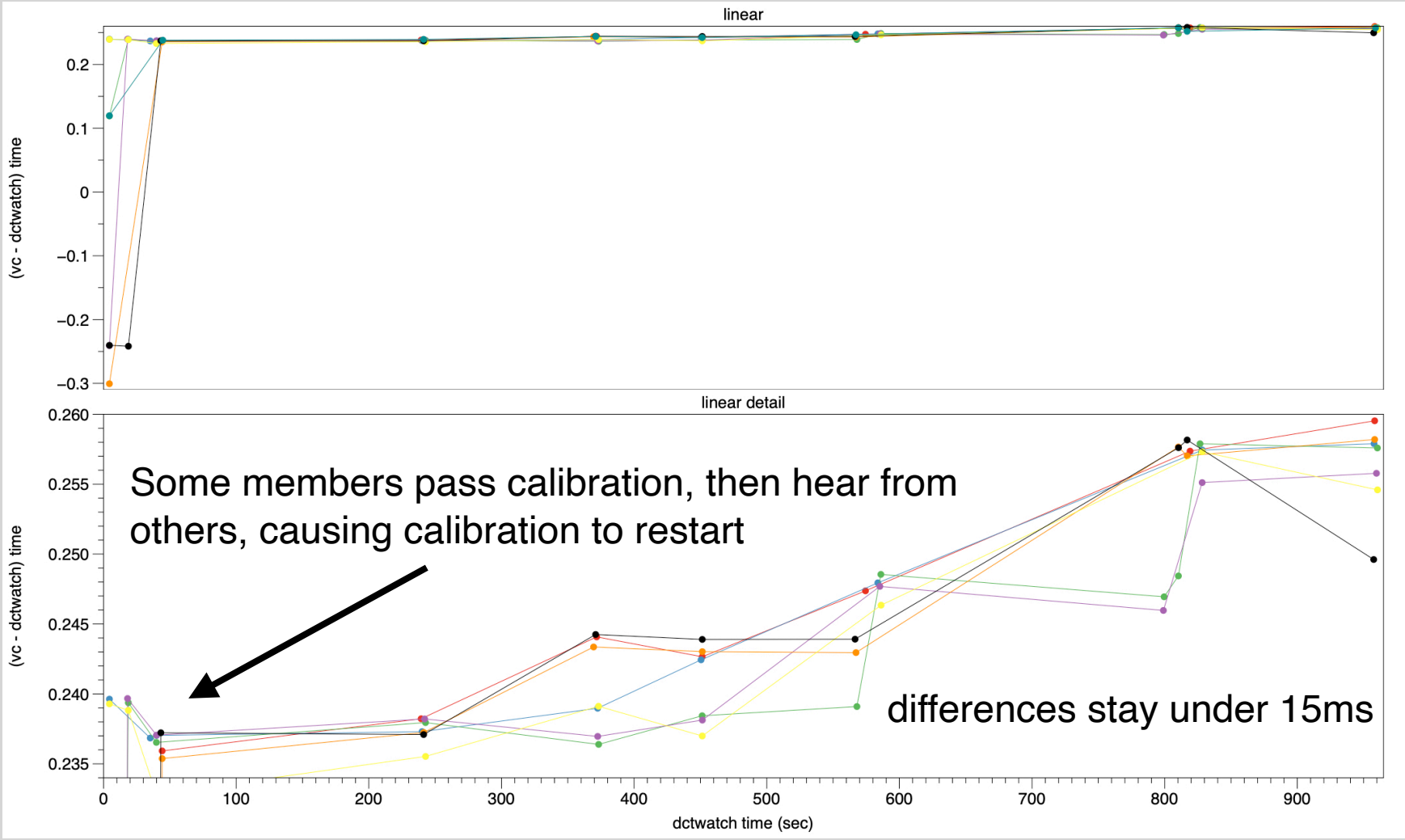

All 8 have the **same** offset after first calibration - the difference (max of 14ms) is just variations in the processing and capture times

Split the members into two overlapping groups (a hack that can be accessed by compiling with the MESHTEST=2 flag)

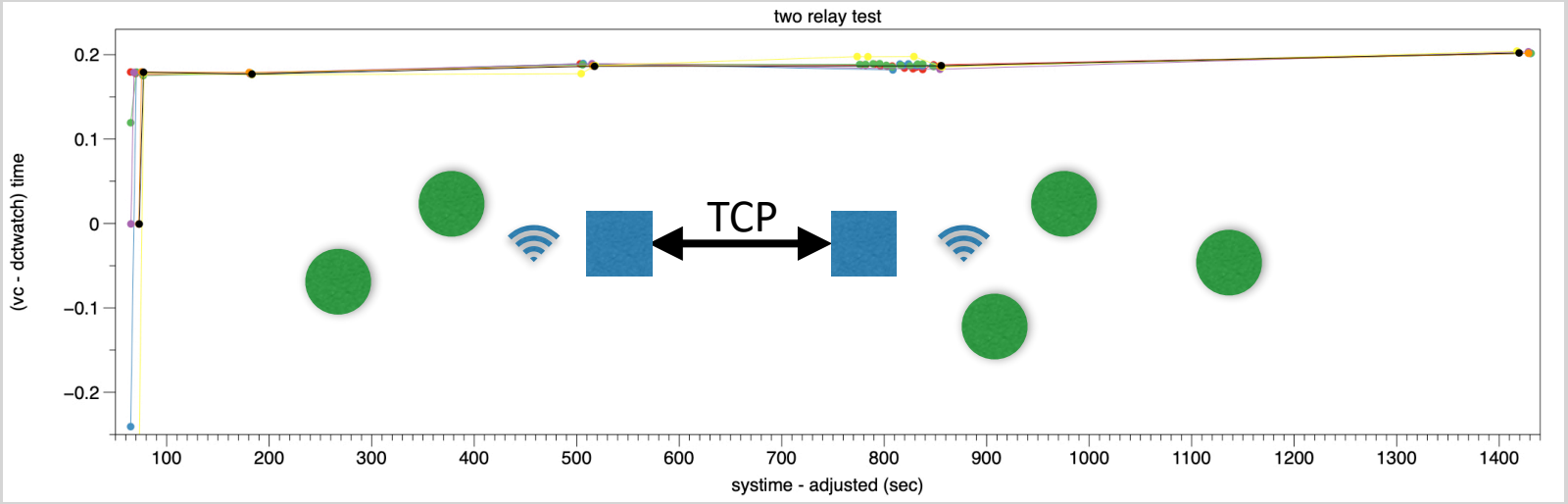


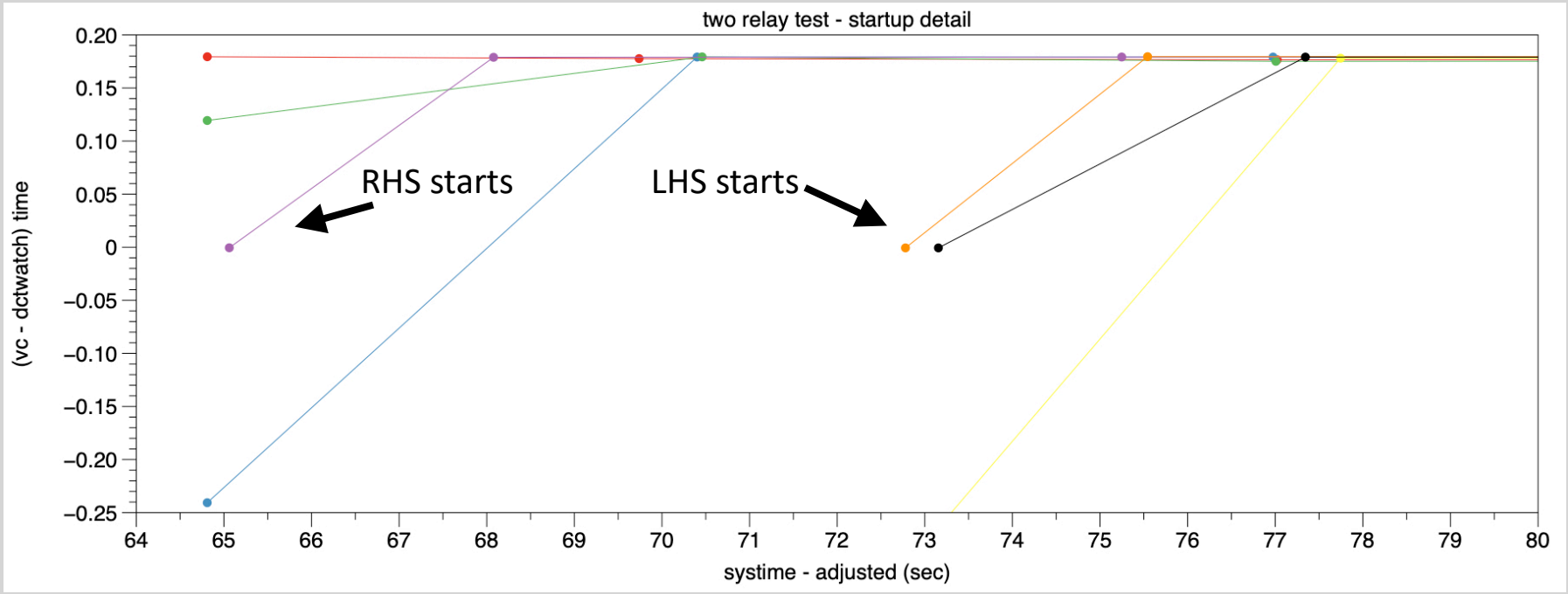



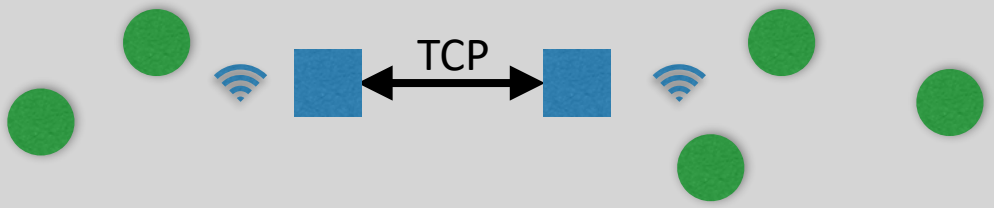
# Next, order member by pid, each talks to adjacent members (MESHTEST=1flag)

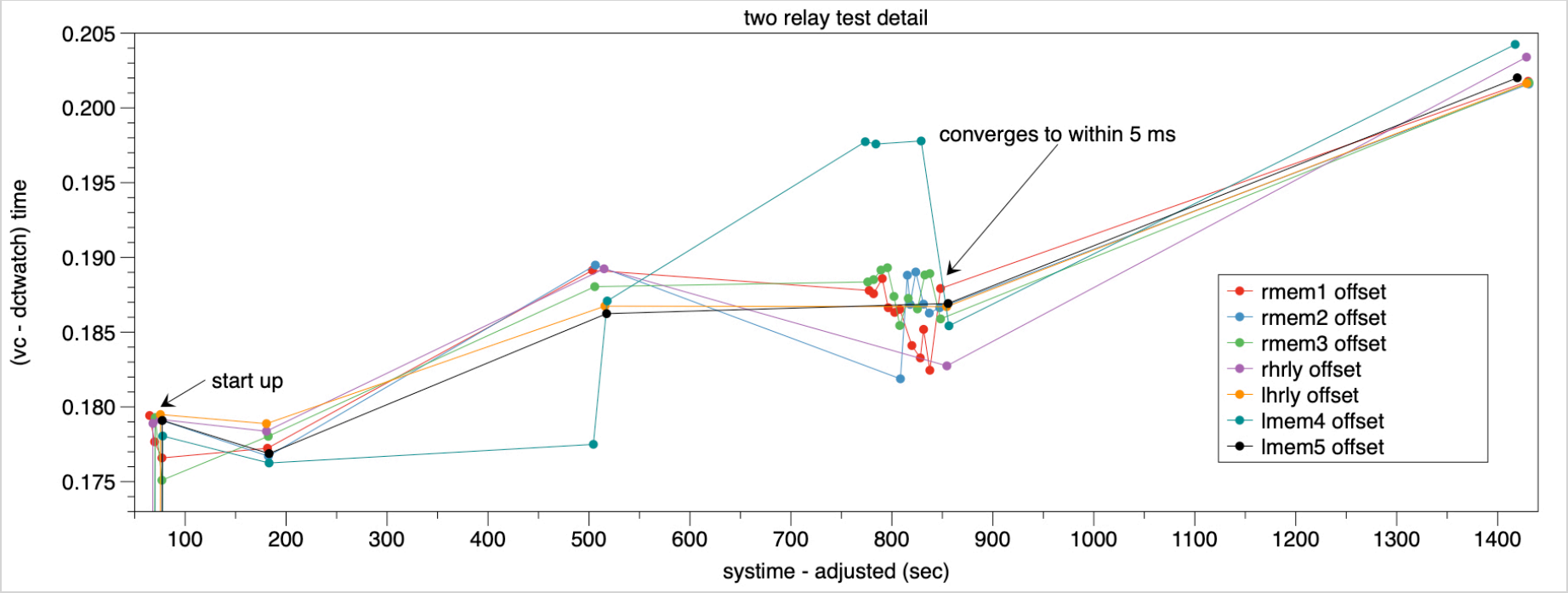


# Connect through relays starting right-side members and relay first




TCP


two relay test detail
0.205
0.200
0.195
0.190
0.185
0.180
0.175
(vc - dctwatch) time
converges to within 5 ms
start up
rmem1 offset
rmem2 offset
rmem3 offset
rhrly offset
lhrly offset
lmem4 offset
lmem5 offset
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
systime - adjusted (sec)

# Next steps

- Larger scale testing will be coordinated by Operant Networks
  - investigate convergence properties
  - investigate robustness to connectivity changes
  - investigate parameters
- Integrate the (existing) round-trip delay computation into determination of quantization and tolerance values
- Integrate use of clock **capability** in identities for devices with access to accurate clocks
- Iterate core algorithm if needed.