First Assignment

Antonio Bucchiarone

Fondazione Bruno Kessler, Trento – Italy
bucchiarone@fbk.eu

04 November 2020

Context

- Interplay between a *data structure*, the append-only log, and *the broadcasting communication abstraction.*

- **Context:** *Secure Scuttlebutt (SSB) project*
  - *https://ssbc.github.io/scuttlebutt-protocol-guide/*
  - *https://handbook.scuttlebutt.nz/*
  - *https://github.com/ssbc/ssb-server*

- An overlay network that is tailored for decentralized social applicat*ions*

- Networking with arbitrary data packets could be replaced by networking with coherent data structures.
- *What are suitable data structures?*
- *What would corresponding networking primitives look like?*

Information Dissemination

- Propagation of *perturbations* is the basis of *information dissemination.*

- A *wavefront* is carrying information omnidirectionally (wireless, water surface), or directionally (in a wire or fiber).
- In **computer networks**, the unidirectional style has become the dominant and default communication model.
  - Data packets typically have a *source* and a *destination*.
  - *Routing algorithms* are graph-based, link-oriented, without exception.
- **Broadcast-centric world**
  - There is no destination, just a source
  - Wireless transmission is not a link but naturally implements broadcast.
  - Routing is not required because perturbations just propagate infinitely far if not blocked.

Broadcast Abstraction

- Broadcast would not the better *base level* abstraction on which to build other communication services.
- Broadcast model and Data Structure (append-only logs).
- Looking at communication problems from a data structure point of view is an important step towards a better understanding of reliable, synchronized, secure, and privacy-preserving operations of distributed applications.
- *Broadcast abstraction* → **solitons** i.e., *information packets that propagate as a solitary wave.*

Broadcast Abstraction

- Broadcast would not the better *base level* abstraction on which to build other communication services.
- Broadcast model and Data Structure (append-only logs).
- Looking at communication problems from a data structure point of view is an important step towards a better understanding of reliable, synchronized, secure, and privacy-preserving operations of distributed applications.
- *Broadcast abstraction* → **solitons** i.e., *information packets that propagate as a **solitary wave**.*

Local and Global Solitary Waves

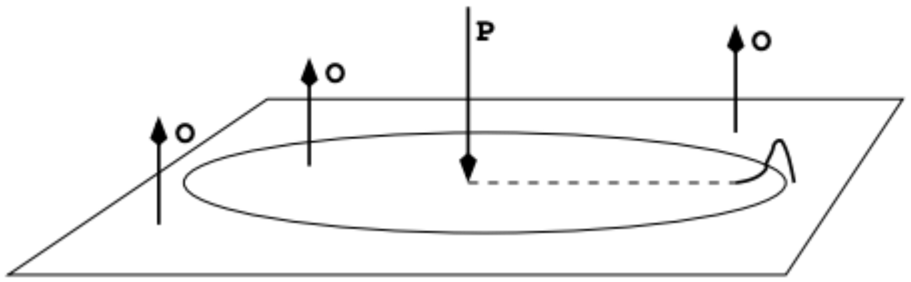
- **Solitons** are *particle waves* or *wave packets* which travel through space without leaving any disturbance behind them.

- To take *solitons as the inspiration for a communication model*.

- Initial perturbation that propagates in a wave-like form, in all directions.
- **Broadcast model properties**:
  1) Each perturbation source has a globally unique identifier that is carried with each perturbation it triggers.
  2) A perturbation and its value eventually reaches all anonymous observers.
  3) All observers sense subsequent perturbations coming from a specific source in the same order.

Broadcast-only Communication Model

- Reliable, ordered broadcast under and asynchronous communication model.



- One **source** *P* triggers a *solitary wave* that some **observers** *O* already have sensed while others still wait for its arrival.

Global Solitary Waves

- Based on the forwarding of local solitary waves, assuming a static network topology and fixed transmission delays.

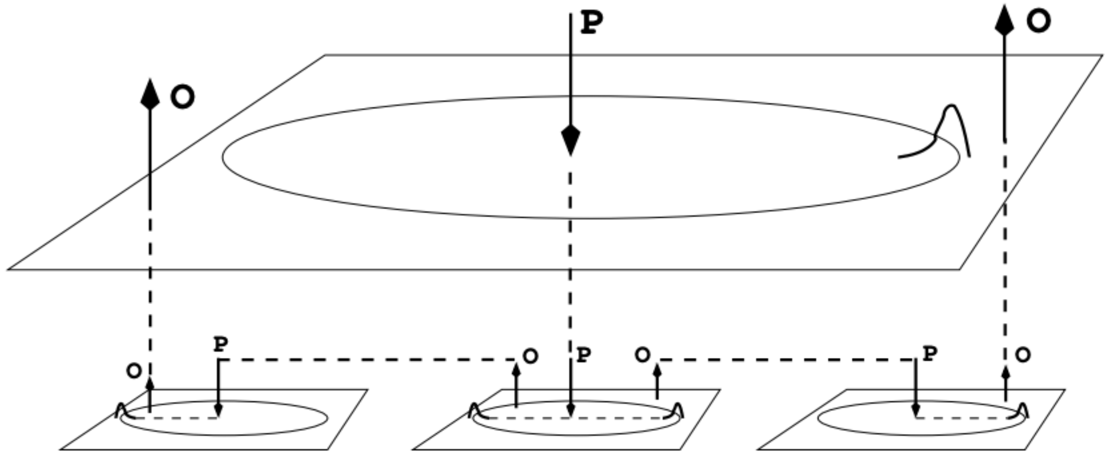
- The **space** is covered with observers called **relays**, capable of picking up solitary waves in one media and propagating them to another media.

- In a physical implementation these media could be the same where the relay just re-broadcasts an amplified version of the sensed wave.

- **Goal:** to build a global broadcast system relying on the concatenation of many local broadcast domains.
- The **Problem** we have to solve is the enforcement of a single **”frontier”** although a perturbation can reach an *observer* via different paths or from multiple layers, hence a different points in time.

Global Solitary Waves

- *Global solitary wave* system using flooding.
- A *local sensed perturba*tion is (immediately or with some delay) forwarded by every neighbour relay.
- **Common case**: A relay senses a perturbation several times, but the perturbation should only be processed once per each observer.
- The need of a duplicate suppression mechanism: also for terminating the global wave instead of having it runs in circles.

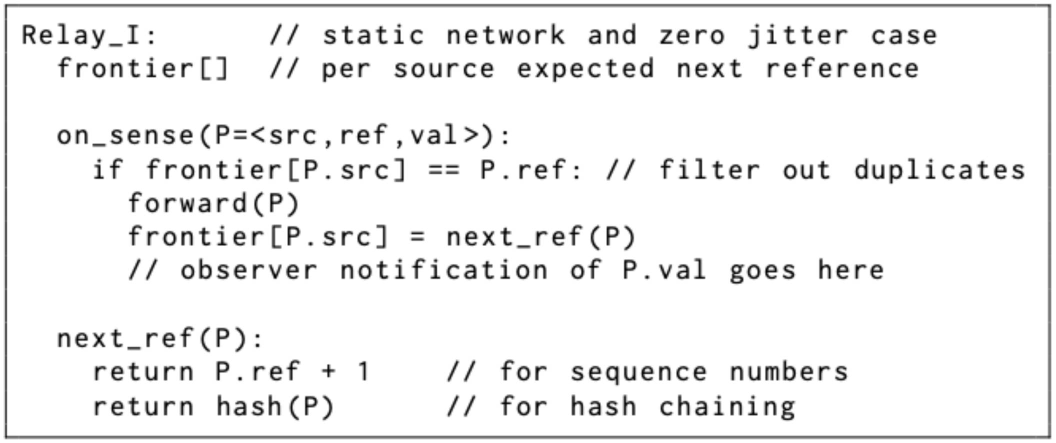


Global Solitary Waves

- Instead of just assuming the total order property (3), we add to each perturbation a reference ref that permits to compute the order or two perturbations.
- **Ex1.:** per-source logical clock that advances for each triggered perturbation.
- **Ex2:** We can include in a perturbation the hash value computed over the representation of the previous perturbation – hash chain.
- Total order of perturbations for each source.
- **Perturbation** as a tuple <*src, ref, val*>

First Implementation

- A **Relay** that bridges two media assuming:
  1) A *static network*
  2) A *constant propagation times*.
- The relay will keep a frontier array of references per sources.
- These variable keep track of the next expected valid perturbation references.



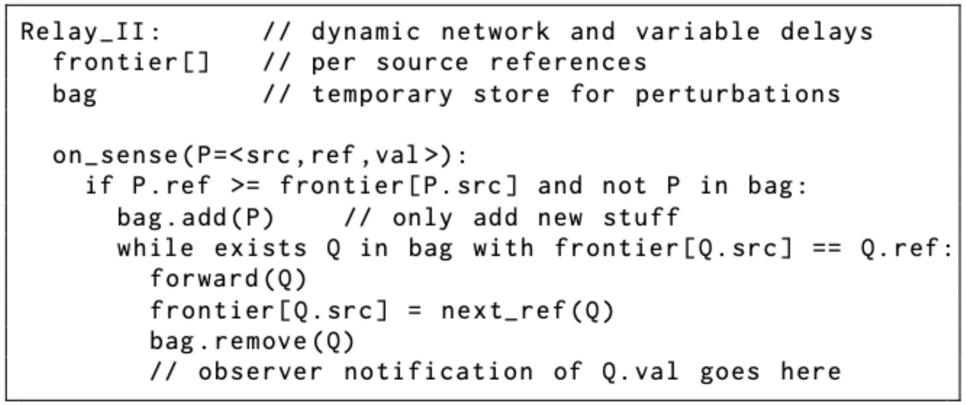
```
Relay_I:        // static network and zero jitter case
  frontier[]  // per source expected next reference

  on_sense(P=<src,ref,val>):
    if frontier[P.src] == P.ref: // filter out duplicates
      forward(P)
      frontier[P.src] = next_ref(P)
      // observer notification of P.val goes here

  next_ref(P):
    return P.ref + 1      // for sequence numbers
    return hash(P)        // for hash chaining
```

Dynamic Network case

- Adding a new relay can instantly lead to a shorter forwarding path such that a perturbation <*src, N*+2, *x*> is potentially sensed earlier than perturbation <*src, N*+*1, y*>
- Ex: Carried over a longer forwarding chain.
- In these cases, a simple counter or hash reference is not sufficient anymore.
- Relays must pay the “price of asynchronicity” in form of memory, typically bound by the maximum jitter.

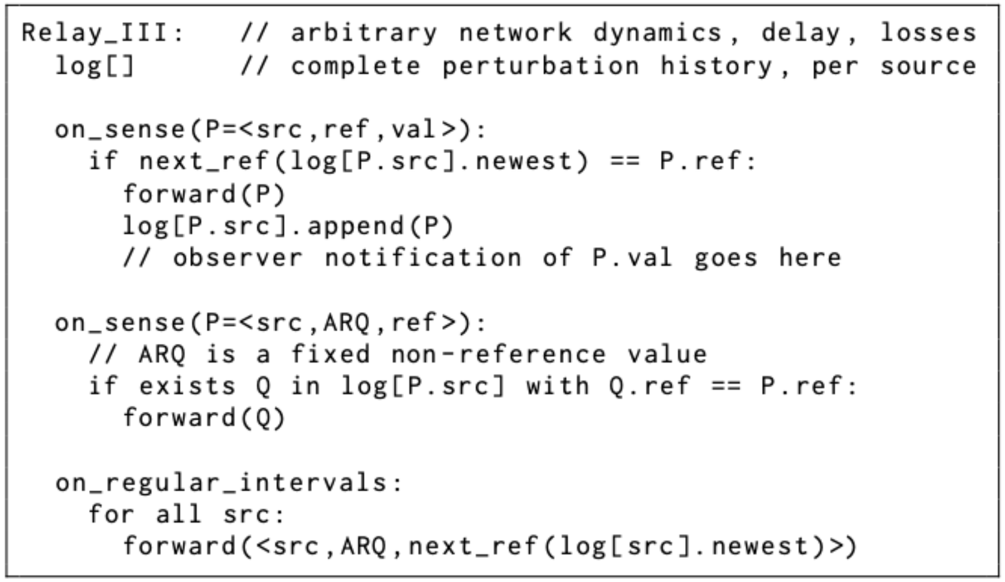


```
Relay_II:       // dynamic network and variable delays
  frontier[]    // per source references
  bag           // temporary store for perturbations

  on_sense(P=<src,ref,val>):
    if P.ref >= frontier[P.src] and not P in bag:
      bag.add(P)     // only add new stuff
      while exists Q in bag with frontier[Q.src] == Q.ref:
        forward(Q)
        frontier[Q.src] = next_ref(Q)
        bag.remove(Q)
        // observer notification of Q.val goes here
```

Recovering from Loss

- **Relays** must implement some *automatic retransmission mechanism*, or use forward error correction and rateless coding.
- A pool mechanism for missed perturbations that complements the (push) broadcast of regular perturbations.
- Relays periodically announce their next expected reference, asking all neighbour to (re-) send them any frontier extension ASAP.



```
Relay_III:   // arbitrary network dynamics, delay, losses
  log[]      // complete perturbation history, per source

  on_sense(P=<src,ref,val>):
    if next_ref(log[P.src].newest) == P.ref:
      forward(P)
      log[P.src].append(P)
      // observer notification of P.val goes here

  on_sense(P=<src,ARQ,ref>):
    // ARQ is a fixed non-reference value
    if exists Q in log[P.src] with Q.ref == P.ref:
      forward(Q)

  on_regular_intervals:
    for all src:
      forward(<src,ARQ,next_ref(log[src].newest)>)
```

Point-to-Point Communication

- We can assign to each communication party a unique source ID that is also used as a destination ID.
- Senders can target specific destinations by generating, for a send(dest,msg) call, a val=<dest,msg> perturbation which will eventually end up in all observers log replicas as <src,ref,<dest,msg>>.
- The observer then watch for new log extensions and filter out the relevant ones.

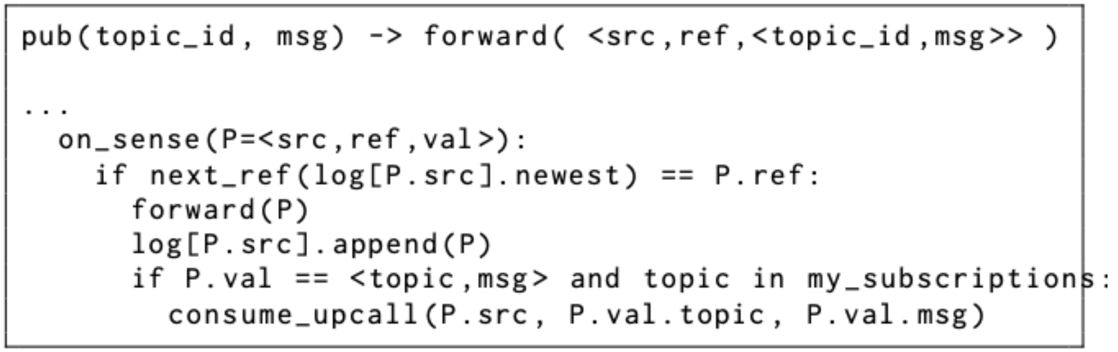
```
send(src,dest,msg) -> forward( <src,ref,<dest,msg>> )

...
  on_sense(P=<src,ref,val>):
    if next_ref(log[P.src].newest) == P.ref:
      forward(P)
      log[P.src].append(P)
      if P.val == <dest,msg> and dest == my_ID:
        recv_upcall(P.src, P.val.dest, P.val.msg)
```

Multicast, Group Communication and Pub/Sub

- One can assign unique IDs to different (multi-cast) groups, or
- Different topics of a pub/sub system

- Observers then compare these destination values with their internal set of destination that encode whether they are:
  - A member of a specific group, or
  - Have subscribed to some pub/sub channel

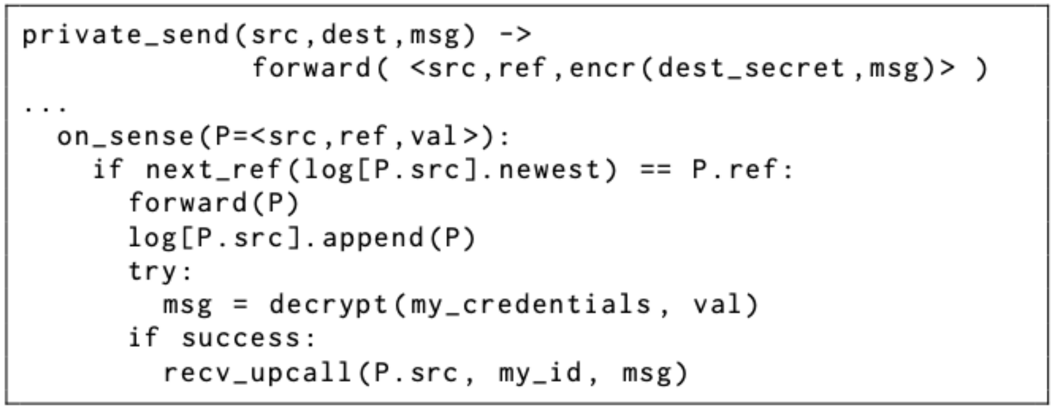


```
pub(topic_id, msg) -> forward( <src,ref,<topic_id,msg>> )

...
  on_sense(P=<src,ref,val>):
    if next_ref(log[P.src].newest) == P.ref:
      forward(P)
      log[P.src].append(P)
      if P.val == <topic,msg> and topic in my_subscriptions:
        consume_upcall(P.src, P.val.topic, P.val.msg)
```

Privacy-Preserving Communication

- To cover fully privacy-preserving communication by encrypting the message.
- Only the intended receiver(s) can decrypt it.



```
private_send(src,dest,msg) ->
             forward( <src,ref,encr(dest_secret,msg)> )
...
  on_sense(P=<src,ref,val>):
    if next_ref(log[P.src].newest) == P.ref:
      forward(P)
      log[P.src].append(P)
      try:
        msg = decrypt(my_credentials, val)
      if success:
        recv_upcall(P.src, my_id, msg)
```

First Assignment

- **Deadline:** Monday 30, November - 6pm
- **References papers available on my website**

1) **PDF document** reporting all the computational analysis of the implemented algorithm using the Simulator and a small description of the model designed (i.e., a tutorial to execute the model)
2) **A GitHub Repo** containing :
    i. a README file that describes the members of the project and a summary of the implemented algorithm.
    ii. Source code of the simulation.
    iii. A JAR file of the model installer.

First Assignment

1) Assuming sufficient bandwidth over-provisioning, this would serve as a baseline for measuring the best latency that can be obtained (for a certain topology and input load).

2) A next simulation variation would then be to continuously reduce link bandwidths and see the effect, go as low as the minimum bisection bandwidth for the given input load, and look again at the latency (before it goes infinite).

3) Un-informed global broadcast will show inefficiency because of redundant dissemination, which would be worth demonstrating and measuring.