① Introduction

• How to sync content across systems?

Content

1. Problem formulation
2. Gossip protocols
3. Network coding
4. Fountain codes
5. De-sync

① Problem:

- Befor: Transfering files reliably
  but if we want synchronize content across systems
- Real problem example:
  „XEROX": maintain a distributed database, all replicas should be coherent (latest and consistent data)

② Initial Approaches

1. Initial: a) Local Updates: updates happened locally
   b) Update Propagation
   c) Method: spread via email
   d) Problems: #3 (run in a background to recover automatically)
2. Anti-Entropy: simple epidemics. [Randomly contact sites]

def: entropy - tendency of system to become increasingly disordered and equalized

- keeping system ordered (low H) requires energy
- we have 2 types of nodes:
  (S) - need 1 or more update and (I) - has at least one recent update
- if start with one I after $\log_2(n) + \ln(n) + O(1)$ (with *push*) we will affect the entire population using $O(n \log n)$ messages
- depending on the design we have 3 operations, in each we compare entire DBs:
  (choose to contact)
  1. push 2. pull 3. push-pull . p_i - site remain S after i+1 cycle
  $p_{i+1} = p_i\left(1 - \frac{1}{n}\right)^{n(1-p_i)}$ (p_i - susceptible) ↓ $p_{i+1} = (p_i)^2$ -//-
- Problem: copy the entire DB just for small changes

3. Anti-entropy: Complex Epidemics Rumor spreading [Every small update = rumor]
   - (S) - doesn't know about update, (I) - knows and shares, (R) - knows, not shares
   - $s + i + r = 1$:
     $ds/dt = -s \cdot i$ (s will be infect); $\frac{di}{dt} = s \cdot i - \frac{1}{k}(1-s) \cdot i$; $\frac{1}{k}$ - prob. of removing
     $s = e^{-(k+1)(1-s)}$
   - when to lose enterest?
     - after k unnecess. contacts
     - after 1
     - with $p = \frac{1}{k}$
   - some prob. of errors

·Idea: check for updates using Anti-entropy (infrequent), but spread small update sets with rumors

③ Gossip protocols

a) Thread → Active / Passive

```
Active thread:
  once in each T time units,
  at a random time, do:
    p = selectPeer()
    send mystate to p
    receive hisstate from p
    mystate = update(hisstate)

Passive thread:
  do forever:
    receive herstate from p
    send mystate to p
    mystate = update(herstate)
```

•Requirements:

1. Random peer selection
2. Nodes use only local info
3. round-based communication
4. limited transmission and processing per round
5. same protocol for all

b) Rumor mongering - special case

c) selectPeer()
- full connectivity
- Only local actions

d. Bootstrap:
1. „Sampling service"
2. Wanna tuning?:
   T-MAN (protocol to manage and optimize the network topology)

e. Gossip for Distributed Computations:
- Structured Aggregation
  - Scatter/Gather Algo
- Unstructured aggregation
  - State = current aprox. of result

  Random Select

  update() → $\frac{s_1+s_2}{2}$ / $\sqrt{s_1 \cdot s_2}$ / $\max(s_1, s_2)$

④ Network coding
1. Separation theorem
2. Allow intermediate nodes to perfom coding ope
3. „Butterfly configuration"
4. Problems: topology-awareness + synchronized stream of inputs

⑤ Fountain Codes
1. Broadcasting problem
2. Approach "Data Carousel" + Raptor codes

⑥ De-sync Algo

② Identifires and Network Association

- How devices and users identify themselves and other within a network ?

Background knowledge:

① Problem

- Chicken-and-egg problem (paradox in establishing a shared view of the network)

② Introduction

- Divide and Conquer Approach:

Conventions on Identifying Scheme
↓ examples:
- IP
  - IPv4 – 32 bit
  - IPv6 – 128 bit
- MAC (Assign to NIC,
- Port service number
- DNS (Domain Name System)
- Structure

Hierarchical	flat
- more effective storage	• not scalable
- Ex.: DNS	• Ex. MAC, UUIDs

- Mappings Identifier spaces:
  - DNS • LDAP • White pages

Conventions on Identifier Acquisition Mechanisms
↓
- Request-Allocate (client request → network gives)
- Select-Advertise (client select → network)
  - ↳ Self-Certifying Names (hash, crypto keys)

③ Network association

- Because of node mobility, we need automatic identifier acquisition

Mechanisms:

① Dynamic Host Configuration Protocol (DHCP)
- def
- adv. + dis

② IPv6 Stateless Address Auto Configuration (SLAAC)
1. Generate Link-Local address
2. Host sends Router Solicitation (RS) to „all routers"
3. Routers reply Router Advertisement Messages
4. Host generates an address in that prefix
5. Duplicate Address Detection (DAD)
6. Active address.

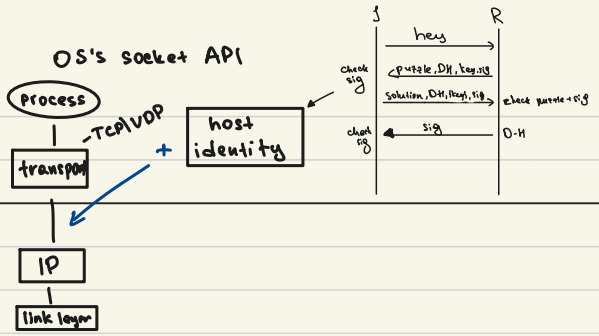
- Prons + cons (only for adress auto-configuration)
- IPv6 Prefix Delegation via DHCPv6
  - a network dynamically obtain a range of IPv6 addresses
  - Using 2 DHCPv6

③ IEEE 802.11 Radio Network Auto Association

④ Multiple identifier schemes in network association

- we can move and switch IPs, but we still we (→ privacy)

① HIP



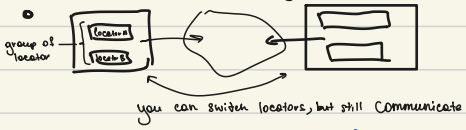
+ RVS for mobile (Rendezvous server)

- LISP / Onions!

② LISP (Overlay IP topology)

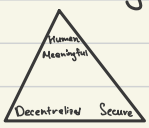
- Edge network → IP = EID • ITR (Ingress Tunnel Router)
- Core network → IP = RLOC • ETR (Egress Tunnel Router)

③ SHIM6 (Site Multi-Homing by IPv6 Intermediation)



⑤ Decentralized Identifiers (DID)

- Use public key
- Let's encrypte
- GNU Name System (GNS)
- Zooko's Triangle



③-④ Cryptography concepts

- How to communicate in private and know with whom ?

① Negligible function

$negl: \mathbb{N} \to \mathbb{R}$, forall $x' > x$

$|negl(x)| < \frac{1}{poly(x)}$

- Asymptotic def (PPT - probabilistic polynomial-time):

> A scheme is *secure* if for every PPT adversary $\mathcal{A}$ carrying out an attack of some formally specified type, and for every positive polynomial p, there exists an integer N such that when $n > N$ the probability that $\mathcal{A}$ succeeds in the attack is less than $\frac{1}{p(n)}$.
>
> Note that nothing is guaranteed for values $n \leq N$.

② Adversary models :

1. Semi-honest (follows, but curious)
2. Malicious (arbitarily deviate)
3. Covert (act maliciously, min. risk of detection)
4. Rational (act based on self-interest)

3. Kerckhoff's Principal

Ex. ax+b with a, b - secret

- Mechanisms must be secure, even if.
  - The algo\protocol is public
  - params are public
  - only random bits used during computations ("keys") can be assumed secret

=> Privacy without obscurity

④ Symmetric Key Encryption

- Idea
-

> **DEFINITION 3.7** *A* private-key encryption scheme *is a tuple of probabilistic polynomial-time algorithms* (Gen, Enc, Dec) *such that:*
>
> 1. *The* key-generation algorithm Gen *takes as input* 1^n *(i.e., the security parameter written in unary) and outputs a key* k*; we write* $k \leftarrow \mathsf{Gen}(1^n)$ *(emphasizing that* Gen *is a randomized algorithm). We assume without loss of generality that any key* k *output by* $\mathsf{Gen}(1^n)$ *satisfies* $|k| \geq n$.
> 2. *The* encryption algorithm Enc *takes as input a key* k *and a plaintext message* $m \in \{0,1\}^*$*, and outputs a ciphertext* c*. Since* Enc *may be randomized, we write this as* $c \leftarrow \mathsf{Enc}_k(m)$.
> 3. *The* decryption algorithm Dec *takes as input a key* k *and a ciphertext* c*, and outputs a message* m *or an error. We assume that* Dec *is deterministic, and so write* $m := \mathsf{Dec}_k(c)$ *(assuming here that* Dec *does not return an error). We denote a generic error by the symbol* $\perp$.
>
> *It is required that for every* n*, every key* k *output by* $\mathsf{Gen}(1^n)$*, and every* $m \in \{0,1\}^*$*, it holds that* $\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m$.

- AES : todo

⑤ Hash function

- def: . Pair (Gen, H)
- Collision: find: $x \neq x'$ $H(x) = H(x')$
- Target - collision : s, x find x' : $H(x) = H(x')$
- Preimage resistant: s, y find $H^s(x) = y$ ← one-way
- SHA256

⑥ Public key Encryption

- Idea
-
- Example: RSA, ElGamal todo

> ***DEFINITION 11.1*** *A* public-key encryption scheme *is a triple of probabilistic polynomial-time algorithms* (Gen, Enc, Dec) *such that:*
>
> 1. *The* key-generation algorithm Gen *takes as input the security parameter* 1^n *and outputs a pair of keys* (pk, sk)*. We refer to the first of these as the* public key *and the second as the* private key*. We assume for convenience that* pk *and* sk *each has length at least* n*, and that* n *can be determined from* pk, sk.
> 2. *The* encryption algorithm Enc *takes as input a public key* pk *and a message* m *from some message space (that may depend on* pk*). It outputs a ciphertext* c*, and we write this as* $c \leftarrow \mathsf{Enc}_{pk}(m)$*. (Looking ahead,* Enc *will need to be probabilistic to achieve meaningful security.)*
> 3. *The deterministic* decryption algorithm Dec *takes as input a private key* sk *and a ciphertext* c*, and outputs a message* m *or a special symbol* $\perp$ *denoting failure. We write this as* $m := \mathsf{Dec}_{sk}(c)$.
>
> *It is required that, except possibly with negligible probability over* (pk, sk) *output by* $\mathsf{Gen}(1^n)$*, we have* $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$ *for any (legal) message* m.

7. Digital Signature
   - Ex. DSA
8. Discrete Log Problem
   - $g^x \bmod p = y$, G-group of prime order p with gen. g
9. D-H key exchange
   - multiplicative group of int, mod N
     number elelements — order
     - elements whose powers cycle through group - primitive generators roots mod N
     - cyclic group - at least 1 generator
     - shared: N-group, g
       privat key: tack 1 number from N
     - $S = g^{privat} \bmod N$
     - $S^{privat} \bmod N$
     - $B^a \bmod N = (g^b \bmod N)^a \bmod N = g^{ab} \bmod N$
     - hard to undo modular exponentiation (discrete log problem)
10. Commitment scheme
    - g, h - generators
    - $g^m h^r = C$
    - (m, r) - commitment
11. Zero-knowledge Proof
    - Schorr Protocol $(h = g^a)$
      Prover: - choose a random $r \in G$
      - $t = g^r$
      - Verifier: - send $c \in G$
      - Prover: $s = r + ca$
      - Verifier: $g^s = g^{r+ca} = g^r \cdot g^{ca} = t \cdot h^c$

$h = g^a$ | $c \in G$
g^b $b \in G$
$d = b + h \cdot c$
$g^d = g^b \cdot g^c$

12. Multiparty computations
    - Preknowledge.
      - def Group $(S, \circ)$ - closure, associativity, identity, inverse)
      - group homomorphism
        $(G, \circ)$ and $(H, \circ)$, $f: G \to H$, $f(g \circ g') = f(g) \circ f(g')$
      - ring homom
        $h(x+y) = h(x) \boxtimes h(y)$ and $h(x \cdot y) = h(x) \boxtimes h(y)$; $h: G \to H$

① Partial Homo Encr:

- ElGamal (·)
  - G, q, g
  - $x \in \{1, \dots q-1\}$ - randomly, private key
  - $h = g^x \Rightarrow (G, q, g, h)$ - public key
  - encryption
    - choose y from $\{1, \dots q-1\}$
    - $c_1 = g^y$ $c_2 = m \cdot h^y = m \cdot (g^x)^y$
    - Ciphertext $E(M) = (c_1, c_2)$
- ElGamal (+)
  - -//-
  - $c_1 = g^y$ $c_2 = g^m h^y$
    $E(M) = (c_1, c_2)$
  - $\frac{c_2}{c_1^x} = \frac{g^m h^y}{g^{xy}} = \frac{g^m \cdot g^{xy}}{g^{xy}}$ ← then brut force for m

G, g, q
$x \in \{1 \dots q-1\}$ - Sk , $y \in \{1, q-1\}$
$h = g^x \Rightarrow (G, q, g, h)$ - Pk
$c_1 = g^y$ $c_2 = m \cdot h^y$ $g^m \cdot h^y$
$\frac{c_2}{c_1^x} = \frac{m \cdot h^y}{(g^y)^x} = \frac{m \cdot g^{xy}}{g^{yx}}$

$$\begin{cases} \cdot \text{decryption} \\ m = \frac{c_2}{c_1^x} = \frac{m \cdot h^y}{g^{y \cdot x}} = \frac{m \cdot g^{x \cdot y}}{g^{x \cdot y}} = m \end{cases}$$

$$E(m_1) \cdot E(m_2) = (g^{y_1}, m_1 h^{y_1}) \cdot (g^{y_2}, m_2 \cdot h^{y_2}) = (g^{y_1+y_2}, (m_1 \cdot m_2) h^{y_1+y_2}) = E(m_1 \cdot m_2)$$

$$E(m_1) + E(m_2) = (g^{y_1}, g^{m_1} \cdot h^{y_1}) + (g^{y_2}, g^{m_2} \cdot h^{y_1}) = (g^{y_1+y_2}, g^{m_1+m_2} h^{y_1+y_2})$$

$s, e - n \bmod q$

$A' \in Nxn \bmod q$

$b = A' \cdot s + 2e$ - public key, s - sk, $A = [b - A']$

En. $C = m + A^T r$

dec $(\langle C, r \rangle \bmod q) \bmod 2$

② Somewhat HE Scheme

- s and e vectors of n random numbers in mod q
- A' - matrix $n \times N$ random numbers
- $b = A's + 2e$
- public key A: $A = [b \; -A]$, secret key: s

→ $C = m + A^T r$ ← distinguishing A from random matrix

$$d = (\langle s, c \rangle \bmod q) \bmod 2 = s(Ar + b) = (sA)r + \langle s, b \rangle = \langle \widehat{e, r} \rangle + \langle s, b \rangle \bmod q$$

(inner product; $e \in [-\frac{q}{2}, \frac{q}{2}]$)

1: $|d| > \frac{q}{4}$; 0 $|d| < \frac{q}{4}$

- Addition

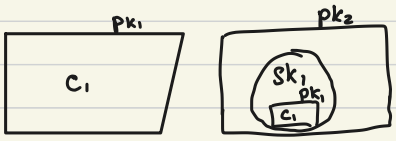
$$c_1 + c_2 =$$

- Multiplication:

$$c_1 \cdot c_2$$

③ Bootstrapping

- SHE with pk and sk
- $c_1 = Enc(m, pk_1)$
- $\overline{sk_1} = Enc(sk_1, pk_2)$



- Recrypt

$$\overline{c_1} = Enc(c_1, pk_2)$$

$$c = Enc(Dec(\overline{sk_1}, \overline{c_1}), pk_2)$$

$$Enc(Dec(Enc(sk_1, pk_2), Enc(c_1, pk_2)), pk_2)$$

- Gentry's bootstrapping theorem

if a SHE scheme can evaluate its own decryption function, then its bootstrappable and can evaluate everything - with no limits

⑤ Information Centric Networking

- What if we want to ask just for data by its unique name?

① Idea

- Location ⟶ name of data

② Named Data Networking

- every piece of data named uniquely; names are hierarchical

  e.g. /ndn/ucla/videos/

- IP <=> NDN (fetch data chunks by names)
  - no adress, only app-def namespaces
  - ndN names data instead of data location
  - consumers fetch data instead of senders pushing packets to destinations

- NDN node's forwarding module

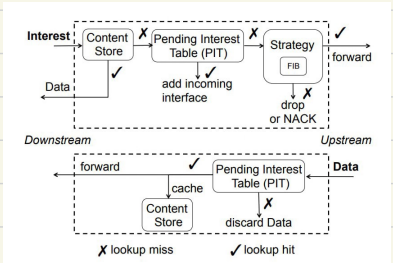


Fig. 3. Forwarding Process at an NDN Node: *upstream* indicates the direction of data producer and *downstream* is to data consumer.

- FIB - forwarding information base
- NDN for. uses the name in Interest (signed Interest)

③ Dona (Data-Oriented Network Arch.) – pull model

- organized around Principals
- flat names <P:L>, P – hash of principal public key, L – label unique chosen by P (hash of object)
- searching by name, Resolution Handlers (RH) name-to-data mapping, similar to DNS
- 3 packet types:
  - find
  - data
  - register

④ Pub Sub (communication model) – push model

⑤ Named Function Networking (NFN)

- get the result, not the data
- Lambda-calculus:
  1. Formal system for the def. and manipulation of functions
     - Effectively computable funct. definable in the lambda calculus
     - Equivalence to Turing Machines
  2. $(\lambda y.\lambda x.(y\ z))\ (a\ b)$

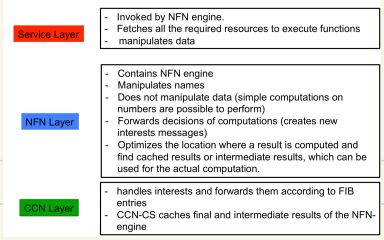
     head: λy — body: $\lambda x.(y\ z)$
  3. call-by-value vs call-by-name vs call-by-need

     $(\lambda x \rightarrow x+x)\ (3+4)$
- NFN: users send block of functions (as lambda)
  - network decides how to: distribute comput, deliver the result, use cashed
  - call-by-name
  - NFN exist in each node

- NFN Node Model

(NDN – Named Data Networking)

- Service Layer
  - manipulates data
  - fetches resources
- NFN Layer
  - NFN engine
  - manipulate names
  - forwards decicions of computation
  - optimizes the locat.
- CCN Layer (Content Centric Networking)
  - handles interests and forwards them according FIB entries
  - CCN-CS cashes final and intermediate results of the NFN-engine



6. Inter-Planetary File System (IPFS) (BitTorrent)
   - DS for storing and accessing files
   - content addressable
   - Peer-to-Peer
   - retrieves data from nearest nodes
7. Secure Scuttlebutt (SSB)
   - decentralized messaging and social networking protocol)
   - content addressable data: each piece of content is signed and append-only list of mes
   - each user hosts their own content and the content of the peers they follow

⑥ CRDT

- How to make all replica strong eventuall consistent ?

① Introduction

- we want to make concurrent transactions, but some of them can be conflicted.
- Database approach
- The blockchain approach
- Replication protocols
  - where
    - primary copy
    - update everywhere
  - when
    - eager replication
    - Lazy replication
  - with Lazy update everywhere we have no consistency

2. Replicas protocol

1. Random Notepad

- Update: 1 step: dice
  - 1-4
    - reconcile with replica 1-4 if not busy
      - 1-3
        - initiator number
      - 4-6
        - responder number
  - 5-6
    - nothing
- Write everywhere:
  - choose [1,9]
  - pick 1 replica
  - replica replace number with client number

↓

- consistency is stable with a non-null probability

- long time to stability
- which state consistent can't be predicted

2 Novelty Notepad Protocol

- Lazy update
- Update: 1 step: dice
  - 1-4
    - reconcile with replica 1-4 if not busy
      - deterministic
        - if tmp > both replica use most recent value
        - keep current
  - 5-6
    - nothing
- Write everywhere:
  - choose [1,9]
  - pick 1 replica
  - Client reads the current tmp of replica, tmp+1 and write new value

↓

- NO, if writes are made on most recent tmp
- yes, most recent writes are concurrent

- may stay inconsistent with concurrent operations

3. The Post-It Protocol

- Lazy update
- Update: 1 step: dice
  - 1-4
    - pull:
      - m>1 you don't own
      - give [1,m-1]
    - push
  - 5-6
    - nothing
- Update Everywhere
  - Client picks one replica
  - [1,9] not already in that replica. write ≥ 4 copies
  - write to replica

↓

- Eventually consistent

- Strong eventual consistency

③ Conflict-Free Replicated Data Types (CRDTs)

- **Strong eventual consistency:**
  1. Eventual update: if an update is applied by a correct replica, then all correct replicas
  2. Convergence: any 2 correct replicas that have updates the same updates are in the same state.
- **Ways to design and reasoning about CRDTs:**

Operation View:

$S_{i+3} = S_{j+2}$ iff any concurrent op_k, op_l are commutative

State view

- $S_{i+3} = S_{j+2}$ iff:
  1. $S_{k+1} = op_k(S_k, \ldots)$ implies that $S_{k+1} \sqsupseteq S_k$ (op_k is monotonic)
  2. $S_{k+1} = S_k \sqcup S_e$ implies that S_{k+1} is LUB S_k, S_e

- **Type of CRDTs:**

Op-based Grow-Only Set

- State: $S_0 \leftarrow \{\}$
- Merge: $S_{i+1} \leftarrow add(add(S_i, x)\ldots)$
- Operations: $S_{i+1} \leftarrow add(S_i, x)$: $S_{i+1} = S_i \cup \{x\}$

- eventually converges as long as all operations are reliably delivered to all replicas.

State-based Grow-only set

- State: $S_0 \leftarrow \{\}$
- Merge: $S_{i+1} \leftarrow S_i \sqcup S_j$: $S_{i+1} = S_i \cup S_j$
- Operations: $S_{i+1} \leftarrow add(S_i, x)$: $S_{i+1} = S_i \cup \{x\}$

- eventually converges even with msm dropped, reordered, duplicated
- requires: 1) eventual delivery of some msm
  2) transitive connectivity between all replicas

- **Partial order**
  - way to define a relationship between elements
  - $S_1 \sqsubseteq S_2 \iff S_1 \subseteq S_2$

- is a binary relation $\sqsubseteq$ between $x_i, x_j \in S$ such that $x_i \sqsubseteq x_j$ is reflexive, transitive and antisymmetric.

- **Monotonic Operations**: $x_{i+1} \leftarrow op(x_i, \ldots)$ is m iff forall values of x_i and arguments $(\ldots)$, $x_i \sqsubseteq x_{i+1}$
- **Merge as Least-Upper Bound (LUB)**
  - at least as large as both S_i and S_j
  - as small as possible

Properties:

- commutative
- associative
- idempotent

④ Designing with State-based CRDTs

1. Set of Possible Values
2. Partial order
3. Monotonicity
4. LUB (Merge)

→ SEC

① Single-Writer Grow-Only Counter

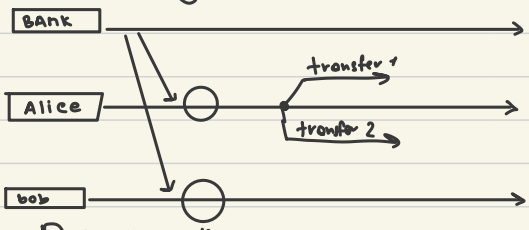
- CRDT design where only one writer that increment the counter.

② Casual-Length Set

- adding + removing elements
- S_i - dictionary, {element: state}
  - if $S_i[k]$ is odd, k in set
  - if $S_i[k]$ is even, k is not in set
- Set of possible states: $KEYS \times [0, \infty]$
- Partial order: $S_i \sqsubseteq S_j$: $keys(S_i) \subseteq keys(S_j)$ and $\forall e\ S_i[e] \leq S_j[e]$
- Monotonic operations:
  - add
  - remove
- Merge (LUB): max

⑦ Double-spending

① Problem

- Double-spending comes from concurrent updates from the same author
- 

  - Detection. this will be eventually detected by correct replica by merging both victims
  - Prevention: restriet state changes, make them sequent.

② Append-Only Log (AOL)

- Sequence of immutable events / msm from a single processor\author.
- In terms of CRDT:
  - Set of possible values: $L_i = ((L_i)_{author}, (L_i)_{last} \in \mathcal{S} = \mathcal{A} \times \mathcal{M}$ such that $(L_i)_{(last)_{author}} = (L_i)_{auth}$
  - Partial Order: $L_i \sqsubseteq L_j$: $L_i, L_j \in \mathcal{S}$ and $(L_i)_{author} = (L_j)_{author}$ and $(L_i)_{last}$ ancestor of $(L_j)_{last}$
  - Monotonic operation:

$$
\begin{aligned}
&L_{i+1} \leftarrow \text{append}(L_i, m) : L_i \in \mathcal{S} \text{ and } m \in \mathcal{M} \\
&\quad L_{i+1} \leftarrow L_i \\
&\quad \text{if } m_{author} = (L_i)_{author} \text{ and } (L_i)_{last} \text{ ancestor of } m \\
&\quad \text{then} \\
&\qquad (L_{i+1})_{last} \leftarrow m
\end{aligned}
$$

  - Merge (LUB)

$$
\begin{aligned}
&L_{i+1} \leftarrow L_i \sqcup L_j : L_i, L_j \in \mathcal{S} \text{ and } (L_i)_{author} = (L_j)_{aut} \\
&\quad (L_{i+1})_{author} \leftarrow (L_i)_{author} \\
&\quad \text{if } (L_i)_{last} \text{ ancestor of } (L_j)_{last} \text{ then} \\
&\qquad (L_{i+1})_{last} \leftarrow (L_j)_{last} \\
&\quad \text{else if } (L_j)_{last} \text{ ancestor of } (L_i)_{last} \text{ then} \\
&\qquad (L_{i+1})_{last} \leftarrow (L_i)_{last}
\end{aligned}
$$

undefined if both *lasts* are concurren

  - Unique author per log in replica + no concurrency of log replicas, so we can represent the state of a replica as a vector clock
    ex: (alice, 2)

③ Git Commit Graph

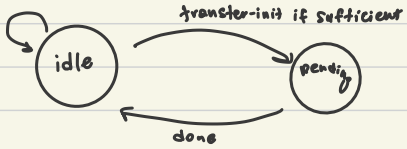
- in append-log there is no casual (how one entry may depend on or relate to another) relationship beetween logs.

④ Secure Append-Only Log

- 2P-BFT-Log
  - problem: totally ordering msm from a given author, similar to an append-only log, while providing eventual consistency and fault-detection in the presence of Byzantine authors that create concurrent msm.
    - 1st phase: growing
    - 2nd phase: shrinking
  - fork ⇏ double-spending. double-spending ⇒ was fork!

⑤ Preventing Double-Spending with Majority Ack Protocol

- Quorum
- To avoid stuck in the pending state forever



- 2PC vs Append-only logs

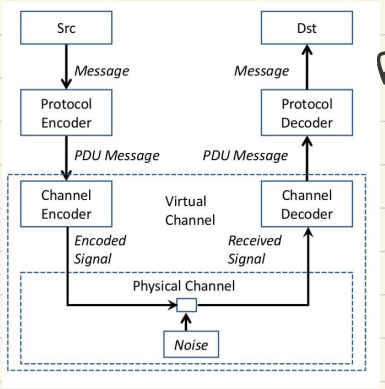
+ two phase

diff: 1. tolerates a subset of Byzantine behaviour
2. no abort, just ignore
3. no ack, just tracking others
4. non-blocking

⑧ Software-Defined Networking

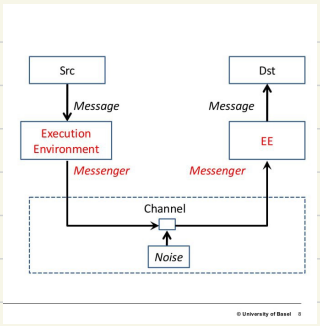
① Active Networking

- Shannon's communication model
- PDU - Protocol Data Unit , PE - Protocol Entity



execution the communication protocol.

PE (fixed number of protocol states)



Idea: want generic environment to compute whatever program it receiver.

So: PE → Execution Environment

PDU → Messenger of Imperative Expression + Data

- Meme concept : Receiver

② Software Defined Networking (SDN)

- Every network device has 3 „planes" of operations

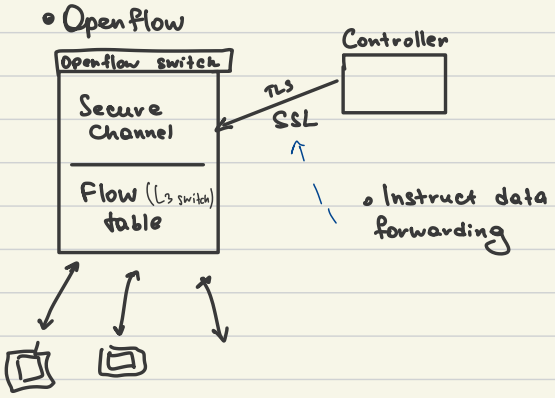
forwarding plane	control plane	managment plane
• directs flow of the traffic	• how data should be forwarded	• configuring, monitoring and maintaining the network device
• FIB: forwarding tables lookup for next hop	• best path • Routhing Path Info (RIB): used to compute the best next-hop to each destination	

- Classic routing protocols in the Internet today

⊕	⊖
distributed and scalable reliability and fault tolerance	Most distributed algo are more complex to implement then their centralized counterparts

- SO: make them a little bit centralized for simplicity for programmability:
  - forwarding → distributed
  - controller → centralized
- We remove the control plane routing protocols from a router:
  result => forwarding by arbitrary flow rule => OpenFlow switch

- OpenFlow



- Flow entry (table):

| match fields | counters | instructions)

↓ | ↓

- IP
- TCP|UDP|port

- APPLY
- CLEAR
- ADD
- Include (metadata)

- NVN

⑩ Bio-inspired

① Introduction

Examples: 1. Bionics 2. Embryonics 3. ANN u. ALife 5. Artificial Chemistry 6. Chemical Computing Models 7. Complex Systems 8.

② Example: The Mate-and-Spread Game

③

- find max => chemistry
- 5 properties of reacting systems:
  1. no distinctions between passive and active
  2. objects can transform into other object without interactions
  3. objects can interact with other objects to create one or more objects
  4. objects can interact in parallel
  5. can regenerate
- def AC: (S, R, A): S-set of molecules, R: set of reactions, A-scheduling algo
- Levels of computations:
  1. Symbolic Chemical Comp. (the result find in a single molecule)
  2. Organizational CC (the result is find in the presence/absence of species)
  3. Dynamic CC (the result in the number of instances of a species)

- Idea of Dynamic CC:
  1. Info: concentration of molecules
  2. State: vector of concentr. of all species
  3. Computation: change in concentration

1. Law of Mass Action (LoMA):

„The average reaction rate (speed) is proportional to the concentration of its reactants"

- $r: \alpha_1 S_1 + \alpha_2 S_2 + \dots \alpha_{|S|} S_{|S|} \xrightarrow{k} \beta_1 S_1 + \dots + \beta_{|S|} S_{|S|}$
- $v = k \cdot \prod_{i}^{|S|} (c_{S,i}^{\alpha_i})$

C- concentration, α_i - consumed by r, β - created by r)

(speed, dynamic)

- Links concentration and rates of a system

- ODEs

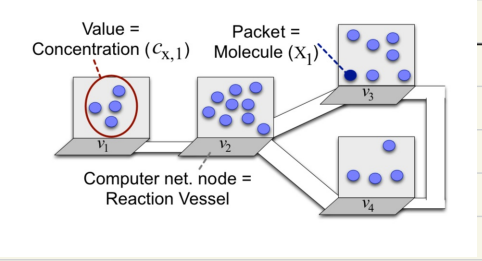
$$\dot{c} = \Xi \cdot v, \quad [\Xi]_{s,r} = \beta_{s,r} - \alpha_{s,r}$$

2. Equilibrium in reaction systems:

1. Law of Mass Action: speed proportional concentration
2. Flow concervation principle: concent- const, speed the same
3. Mass conservation principle: the number of molecules is conserved in reaction loops

3. Stochastic simulation algo

1. Naive
2. Next method



④ Chemical Networking Protocols and Algorithms:

1. Distributed Artificial Chemistry

- Each species is a node in computer network
- Reaction model of communic: reactions consume molecules in one node and produce molecules in other nodes
- Reaction types:
  - Local reactions: local consume, local produce
  - Communicating/distributed reaction: local consume, local\remote produce

2. Gossip-style: Chemical Algorithms

Task: compute an average value in distributed manner

I. Gossip Approach: $S_n = \frac{S_{old} + S_{new}}{2}$

☹ hard protect malicious attacks, not robust

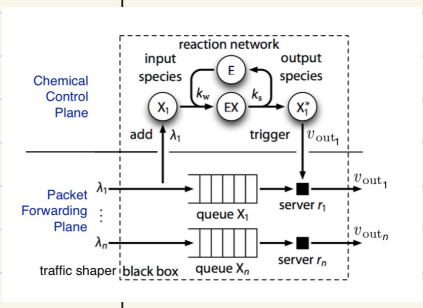
II. Chemical alternative:

Idea: • concentration represent the approx. value
- send molecules from one node to a neighbore node

- Disperser - Chemical Protocol
- Consensus in Wireless Sensor Networks (WSNs)
  - Idea + embodiment concept
  - Nodes end up in a correct state, independently of their initialization + cope with not foreseen faults.

3. Chemical Rate Control Algorithms

1. LoMA as scheduling discipline
2. ChemFlow: Rate limiter/controller



Chemical Control Plane — reaction network

Packet Forwarding Plane

- Enzymatic model
  - E and EX; $C_E + C_{EX} = const = e_0$, $v_{max} = e_0 k_s$
  - Low-pass filtering ($k_s \sim$ cut-off freq.)
- LoMA vs Enzymatic: Run-time programmability

4. Distributed Rate Controller

- Problem
- Idea: look at local load and load at peers