Transactional Support for SDN Control Planes

Petr Kuznetsov
Telecom ParisTech

WTTM, 2015

Software Defined Networking

- An emerging paradigm in computer network management
- Separate forwarding hardware (data plane) from network managing software (control plane)

- Simplify network management
- Facilitate innovation
- Improve QoS

Software-Defined Networking in one slide

A small number of « SDN servers »

SDN controller

rwarding

Load balancing

…

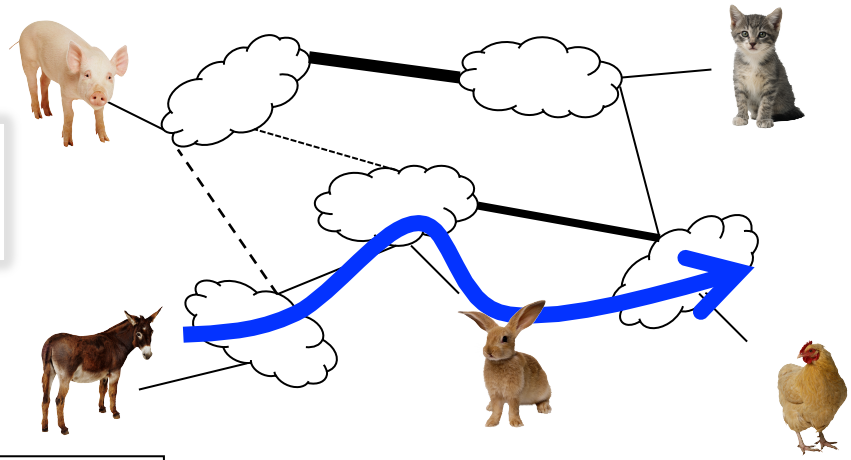
Monitoring

Security

Standard API (OpenFlow)

- Rules (match-action):
  - Match mask -> forward to port A, add tag, drop, update counters …
- Sets of rules (flow tables) spacify a policy

A lot of SDN-enabled switches

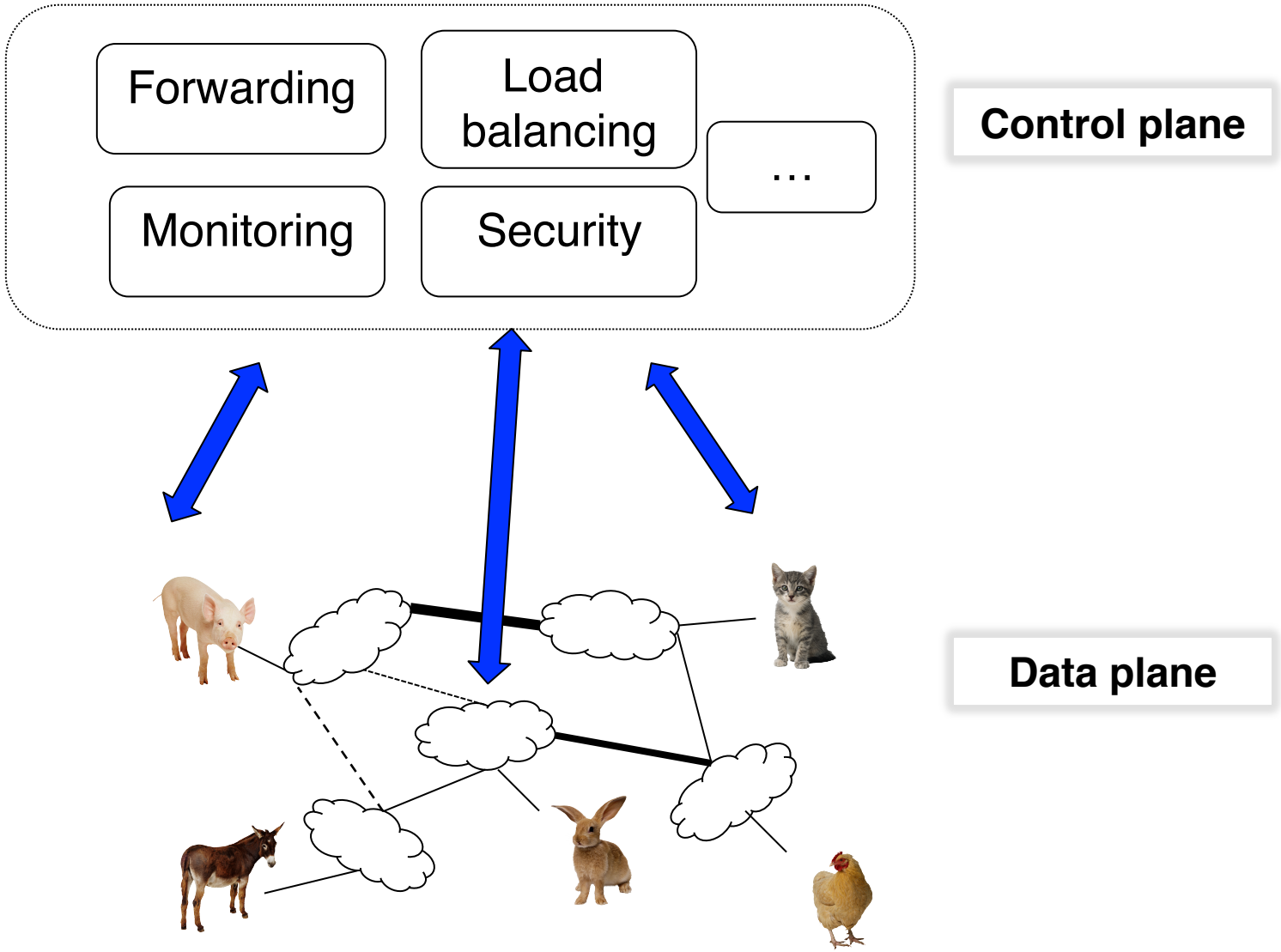


```
forward http from x to y
```

SDN challenges [CACM Oct 2014]

- Network-wide abstractions: consistent snapshot of what is going on: topology, traffic load (e.g., encoded in a NIB)
- Distributed updates: modifying rules at multiple switches while preserving consistency under transition
- Modularity: multiple composition levels, programming interface allowing for composition
- Virtualization: enabling the virtual network abstraction
- Verification: tools for checking the network program correctness

Consistent Network Information Base?



Atomic network snapshot

- Benefits: calculations based on global view of the "network state"
  - ✓Computing shortest paths
  - ✓Load balancing
  - ✓Traffic engineering
- Difficulties: physically impossible to get the state atomically
  - ✓The outcome of the calculation may be imprecise
  - ✓"Double-collect" technique?
  - ✓Read transactions? Abort if "consistency" cannot be achieved

Distributed updates

- Central controller – no need for distributed protocols?
  - ✓Atomic update impossible
  - ✓But can be emulated
  - ✓Sometimes…
  - ✓Hard to do with coexisting data-plane traffic
- Concurrent updates?
  - ✓Necessary for scalability and robustness?

- Challenges
  - ✓Maintain some consistency during the transition period (per-packet/flow consistency, loop-freedom, blackhole-freedom)
  - ✓Compose concurrent updates, reject if not “composable”

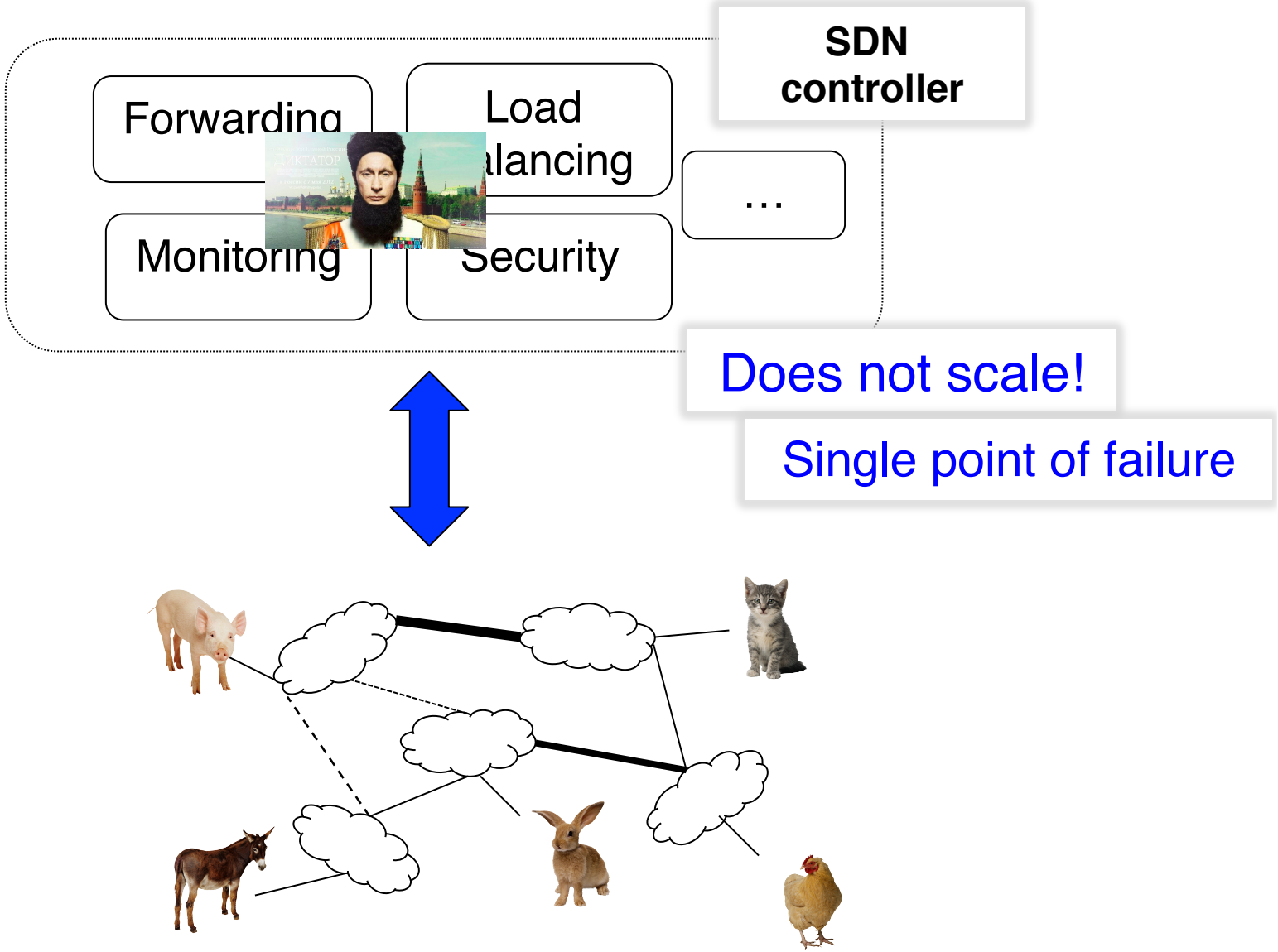
Two-phase update (single controller)

[Reitblatt et al., 2012]

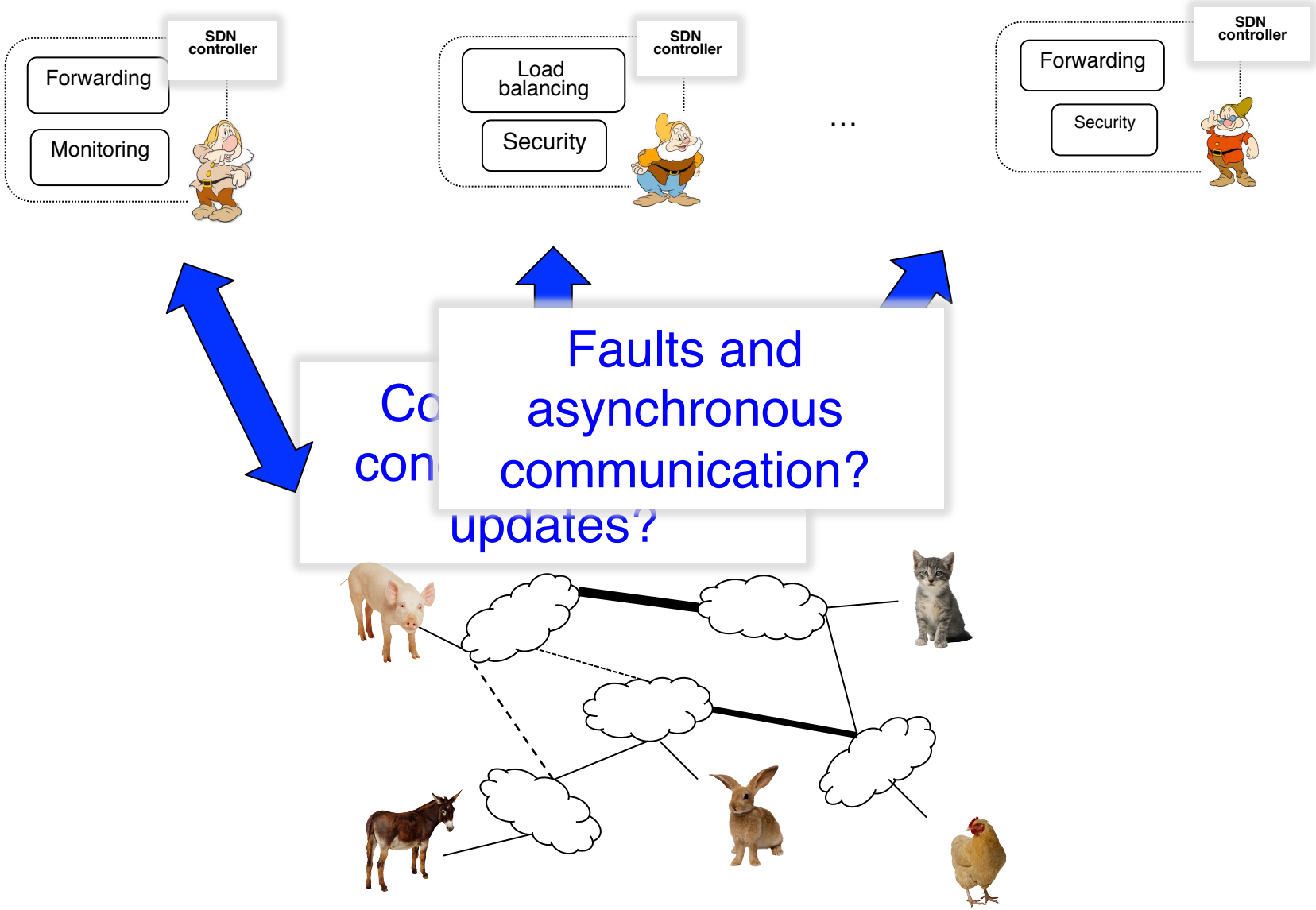
- Choose a unique tag t for the new policy
- Phase 1: update all internal ports with policy rules for the chosen tag t
- Phase 2: update all ingress port to tag incoming packets with t

Per-packet consistency: every packet is processed by the old policy or by the new one

Centralized network control?



Distributed Network Control?



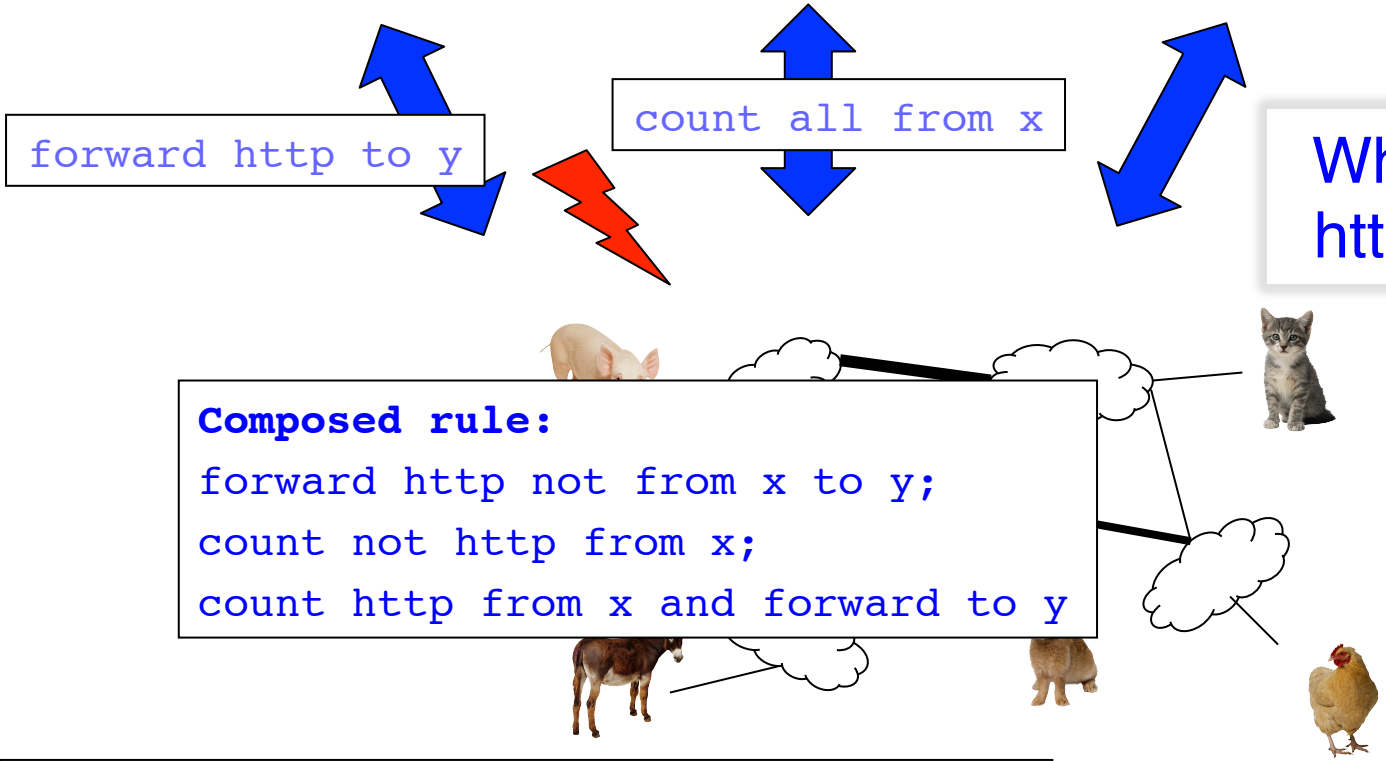
Composing (composable) policies

SDN controller

SDN controller

…

SDN controller



What to do with an http packet from x?

Composed rule:

```
forward http not from x to y;
count not http from x;
count http from x and forward to y
```

Centralized setting: Frenetic, Foster et al., ICFP 2011

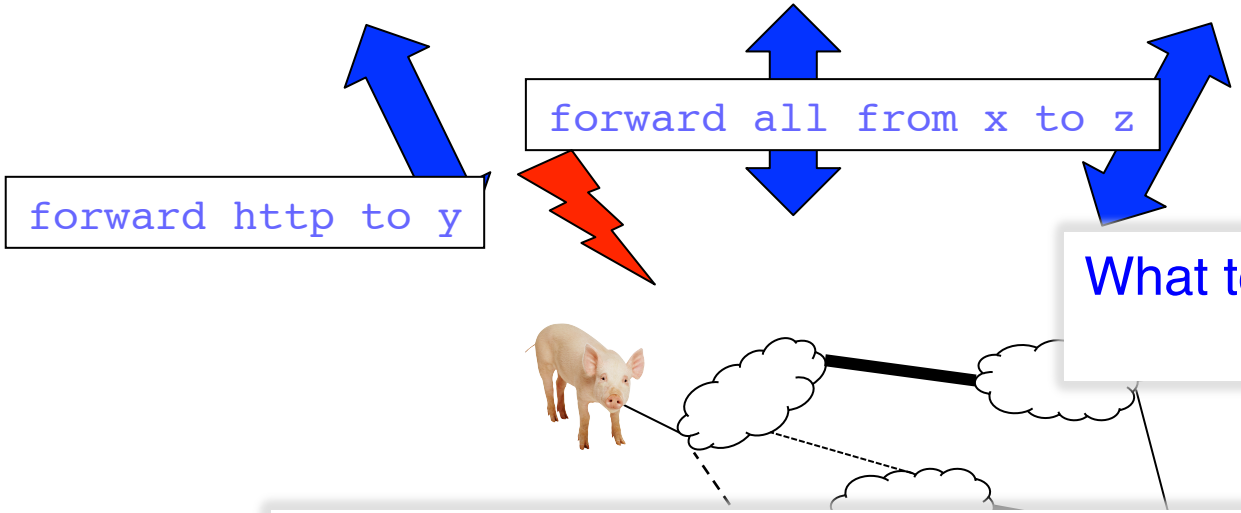
Rejecting conflicting policies

SDN controller

SDN controller

…

SDN controller



What to do with a packet from x to y?

A policy must be completely installed or completely rejected!

Transactional memory?

Software Transactional Memory in one slide

Mark sequences of instructions as an atomic transaction:

```
atomic {
        if (tail-head == MAX){
        return full;
        }
        items[tail%MAX]=item;
        tail++;
}
```

- A transaction can be either committed or aborted
  - ✓ Committed transactions are serializable
  - ✓ Let the transactional memory (STM) care about the conflicts
  - ✓ Ease of programming and efficient (?) use of concurrency

STN: creating the illusion of atomicity

- Each network operation appears as executed atomically
  - ✓Snapshot (read-only)/update (write)
  - ✓If not possible: abort (as it never happened)
  - ✓Even if traffic overlaps with operation intervals
- Sequential composability: for each real history (observable trace), there exists a history S:
  - ✓Sequential: every control operation or traffic trace/flow sequential in S
  - ✓Legal: committed updates compose, snapshots are consistent with the committed past, traffic is processed by the latest committed policy
  - ✓Equivalent: no system component sees the difference from S
- Like serializability for transactions

STN: Interface

1. A controller executes an operation π by invoking `exec(`π`)` that returns `ack` or `nack`

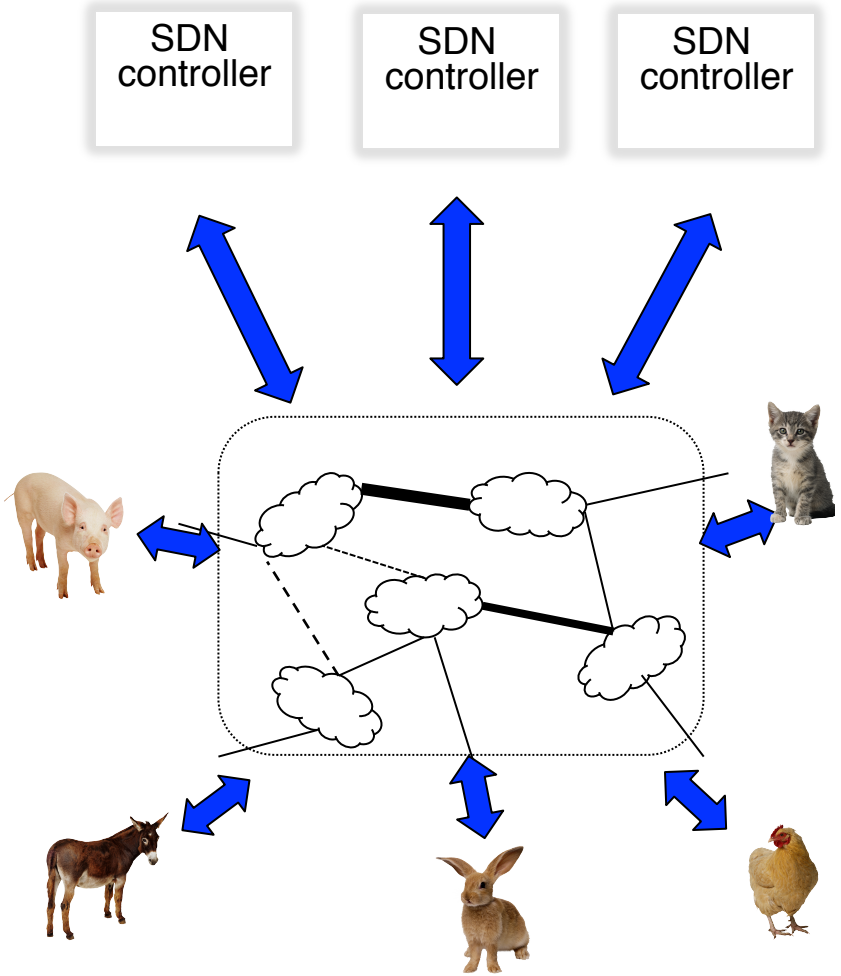
- `res` $-\pi$ is executed (committed) with result `res`
- `nack` $-\pi$ is rejected (aborted)

2. The environment injects traffic via ingress ports:

- `inject(pk,j)`: packet pk enters ingress port j

3. A packet traverses the network according to the current policy:

- `forward(pk,i,pk',j)`: packet pk arrived at port j and forwarded to port j (possibly modified)



STN: correctness

Every history H of satisfies:

- Sequential composability: a *completion* of H (aborted policies ignored) is *indistinguishable* from a legal sequential history S
- Progress: only conflicting policies are aborted (exactly one is committed)
- Liveness: an operation request by a correct control unit eventually returns

Application: Consistent Policy Composition (CPC) [Infocom 2015]

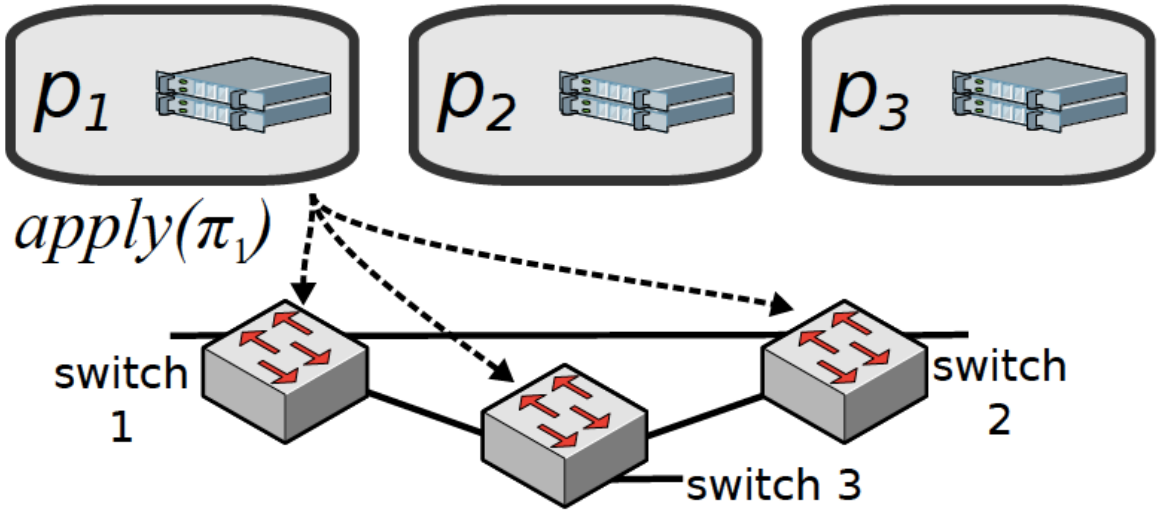
Controllers independently apply policy update requests:

- Conflict-freedom: no two conflicting policies installed
- Progress: non-conflicting policy eventually installed; and: at least one policy commits
- Per-packet consistency: every packet witnesses exactly one policy applied (during its network traversal)

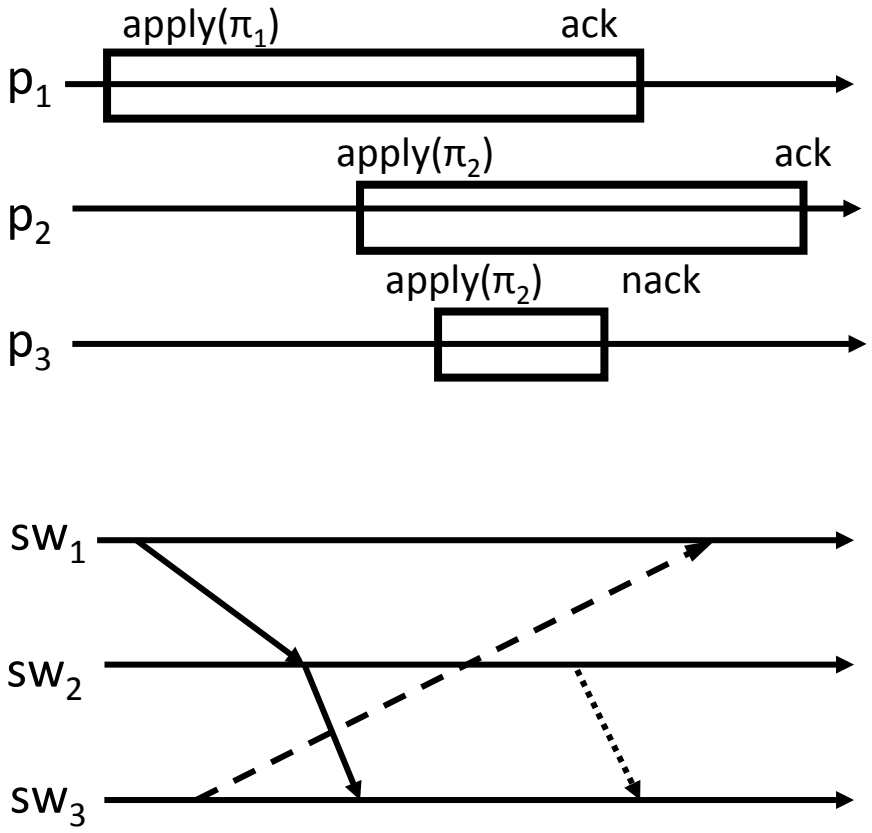
(Traffic is not under our control!)

CPC: Example

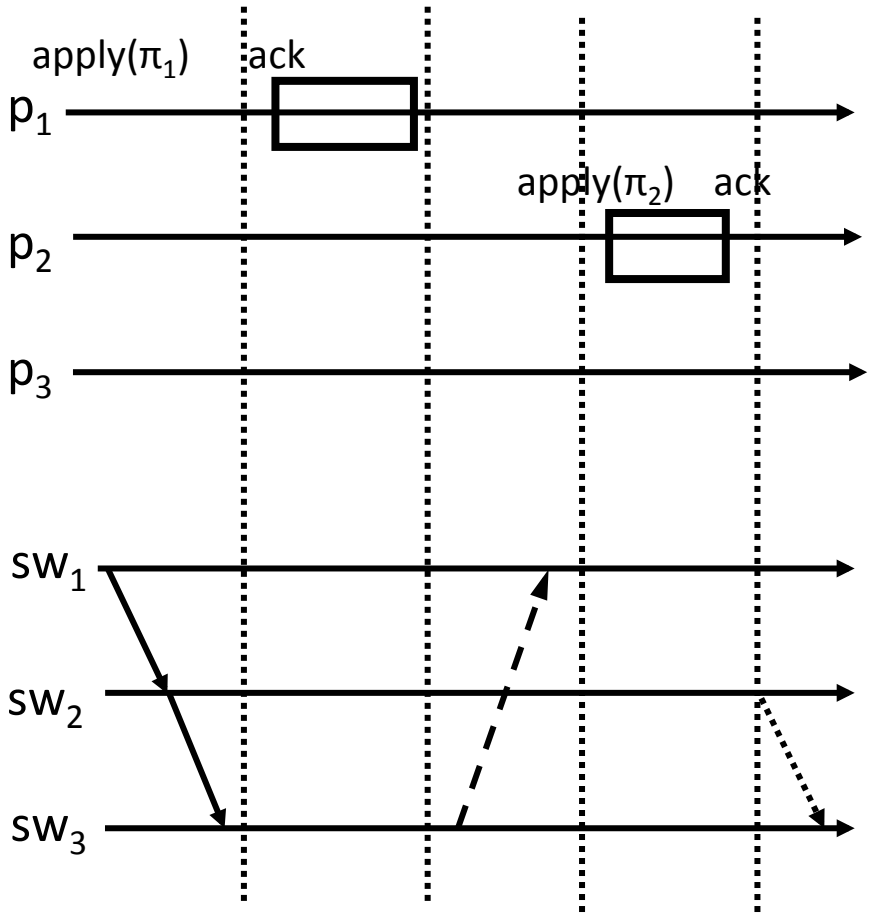
- 3 switches, 3 controllers



CPC: Example



Original history



Sequential equivalent

CPC: TM analog

Two kinds of transactions:

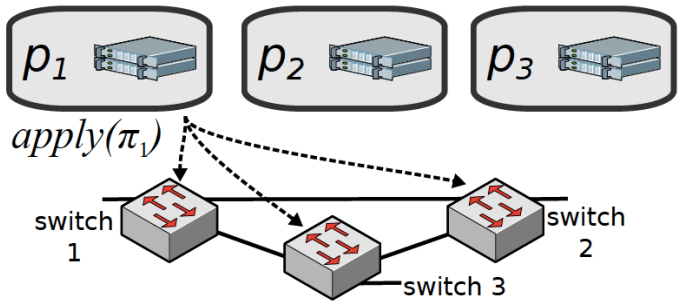
- Policy updates invoked by the controllers
- Located paths injected by the environment

Semantics:

- (Committed) transactions appear sequential
- Policy updates abort when conflict
- Located paths never abort, witness exactly one policy
  - ✓packet arrival and link delays are out of our control
- ≈ MV-permissiveness (read-only txns never abort)
  - ✓Paths are not really read-only

CPC implementation: model v1

- Controllers access ports with read and write ops
- Controllers can communicate via asynchronous message-passing
- Controllers may fail by crashing
- No synchrony assumptions
- Restrict policies to forwarding
  - ✓Compose if domains are disjoint or related by containment
  - ✓Reject otherwise

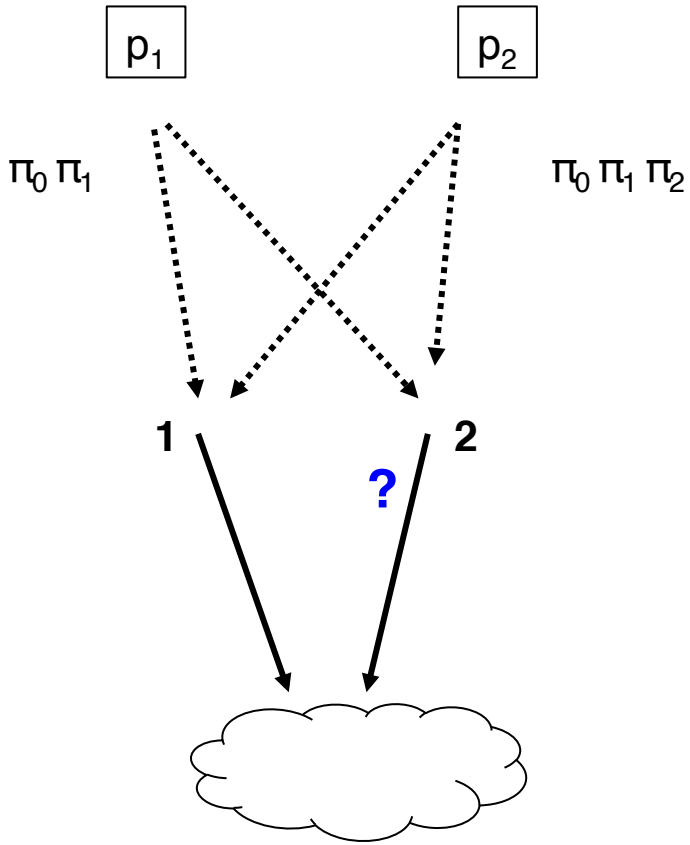


Asynchronous read-write CPC

Theorem: 1-resilient read-write CPC is impossible.

Proof sketch:

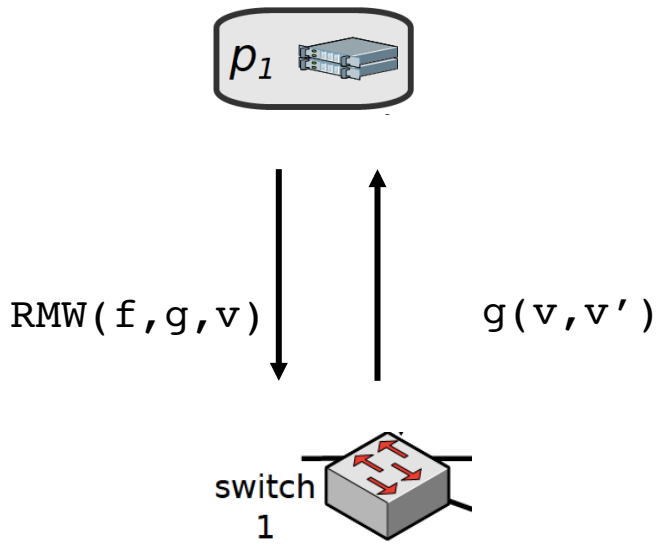
- Two ingress ports 1 and 2 initially forward all to the internal ports (π_0)
- π_1 installed by p_1 and π_2 installed by p_2, π_2 refines π_1 (higher priority, same domain)
- π_1 and π_2 propose different paths
- p_1 changes port 1 and is just about to change 2 (with a composition of π_0 and π_1), p_2 takes no steps
- p_2 wakes up and installs of $\pi_0\ \pi_1\ \pi_2$, p_1 takes no steps
- p_1 changes port 2 with $\pi_0\ \pi_1$: π_2 is forgotten!



CPC implementation: model v2

- Controllers access ports with atomic read-modify-write ops `RMW(f,g,v)`:
  - ✓read the state `v'`
  - ✓write `f(v,v')`
  - ✓return `g(v,v')`

- Intuition: do not update if conflicts with currently installed policy



RMW in Openflow 1.4

- Bundling: multiple control messages organized in mini transactions:
  - ✓All messages in a bundle carry the same bundle id
  - ✓If a single message is rejected, the whole bundle is rejected
- `OFPFF_CHECK_OVERLAP` flag: detect conflicts "on-the-fly"
  - ✓A conflicting rule is rejected
  - ✓A control message is sent if conflict detected
- Cookies: controller-specific modifications of individual flow entries

Upper bound: FixTag algorithm

Operation:

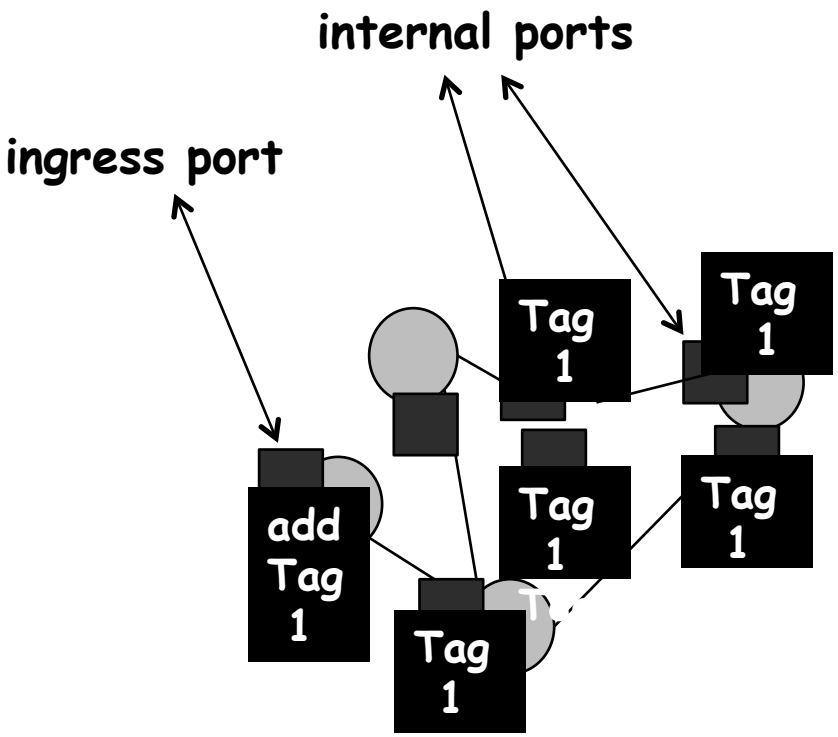
1. Unique tag per policy
2. Install at internal ports first (compose if necessary)
3. Once installed at internal ports…
4. … add the tag to all packets at ingress ports!

Upsides: wait-free (tolerates all failure patterns)

Downsides: overhead can be huge

- Super-exponential in the size of the network

- Internal ports are ready when needed
- A packet is processed by exactly one (composed) policy



Can we do better?

- No, if we get no feedback from the network
  - ✓ Tag t cannot be reused if it is still "in flight"

- Suppose, we can correctly evaluate the set of active tags
  - ✓ Correct (but asynchronous) oracle
- Single-controller scenario: one bit is enough!
  - ✓ Upon policy update π_i wait until (`i mod 2`)-traffic is over, and use tag `i mod 2`

- Two or more controllers: inherent price of concurrency?
  - Between constant and super-exponential?
  - Yes, if controllers are "synchronized"

CPC protocol: ReuseTag

- Mark incoming packets with policy tags
- Proportional to the level of resilience:
  - ✓Up to f failures: f+2 tags needed (proved optimal)
- Controllers use consensus instances (Paxos based on eventual synchrony or «eventual leader»)
  - ✓Replicated state machine for request-tag ordering
- Two-phase update per ordered (composable) request
  - ✓Atomic read-modify-write (RMW) to avoid "overwriting"

ReuseTag properties

- All requests are serialized
  - ✓Even non-conflicting ones
  - ✓Allowing for more concurrency of updates?
- RMW port operations
  - ✓Check a condition on the switch state – update the state if satisfied: OpenFlow?

Summary: software transactional networking

- Control middleware: manipulate the network as though there is no concurrency
  - ✓Conflicts result in aborts or composition
- Run sequential code (in the form of txns)
- Committed transactions and traffic paths appear sequential
- Aborted transactions “do not appear”

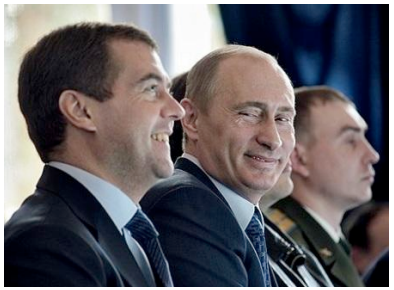
STN for SDN

- Network-wide abstractions and distributed updates: modifying rules at multiple switches while preserving consistency under transition
- Modularity and virtualization: transactions are compositional
- Verification: easier to verify (semantically sequential) transactional programs

New? Interesting?

- New: distributed network control
  - ✓Network races
  - ✓Notions of conflict and consistency

- Interesting: new kinds of concurrency
  - ✓Traffic as an independent thread
  - ✓Not all is about consensus



References

- M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. Software transactional networking: Concurrent and consistent policy composition. ACM HotSDN, August 2013

- M. Canini, P. Kuznetsov, D. Levin, and S. Schmid. A Distributed SDN Control Plane for Consistent Policy Updates. INFOCOM 2015 (check arxiv)

- Liron Schiff, Stefan Schmid, Petr Kuznetsov. In-Band Synchronization for Distributed SDN Control Planes. In submission.

Communities?

- [Kreutzh et al., Software-Defined Networking: A Comprehensive Survey, 2014]: «distributed» appears 73 times
- in 407 references starting roughly in 2010:
  - ✓61 SIGCOMM
  - ✓34 HotSDN
  - ✓50 HotNets
  - ✓20 CCR
  - ✓25 NSDI/OSDI/ATC

  …
  - ✓1 PODC
  - ✓0 DISC

Eskerrik asko!

Questions?