Toward Synthesis of Network Updates

Andrew Noyes – Cornell University

Todd Warszawski – Cornell University

Pavol Cerny – University of Colorado Boulder

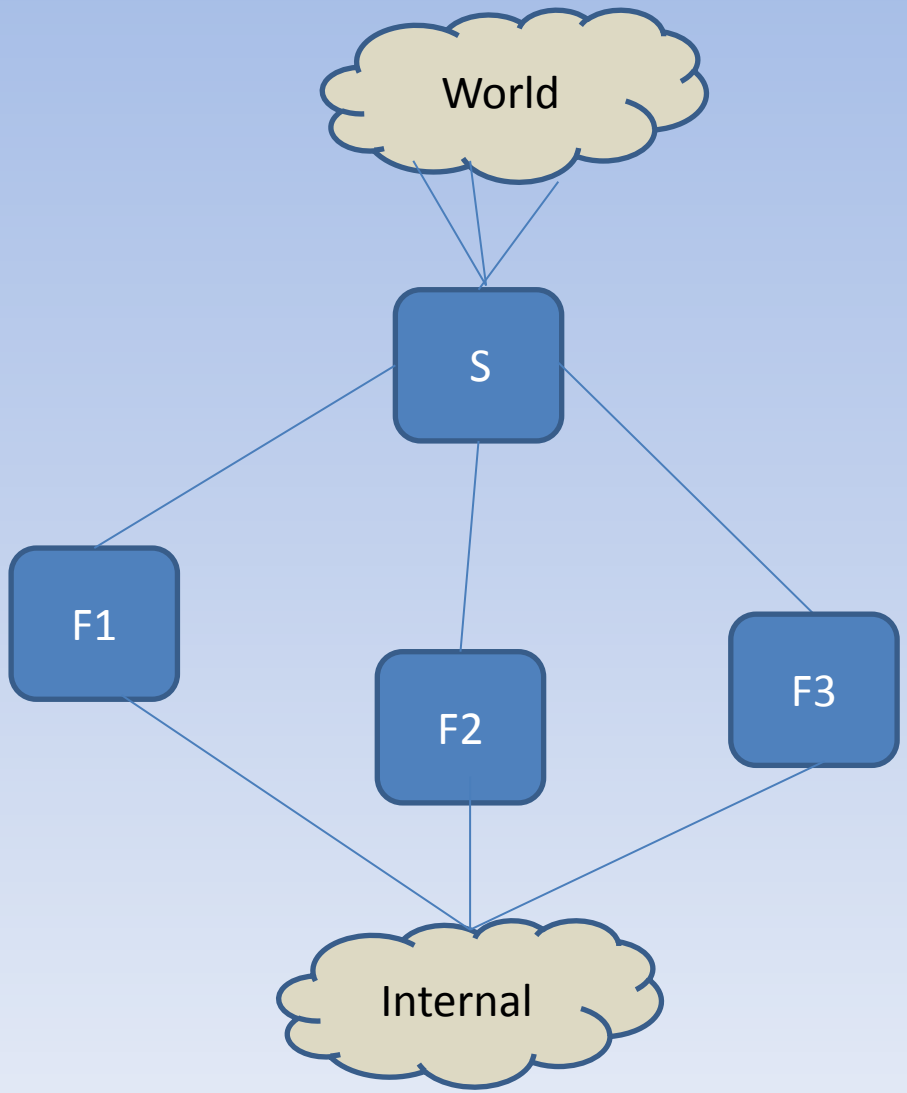
Nate Foster - Cornell University

Presented by: Ronen Jacobi.

Introduction

- Most networks are updated frequently.
- Implementing network updates while traffic continues to flow is very difficult.
- Bad implementing can cause severe damages.
- Mechanisms that were invented are too general and hard to implement or very limited

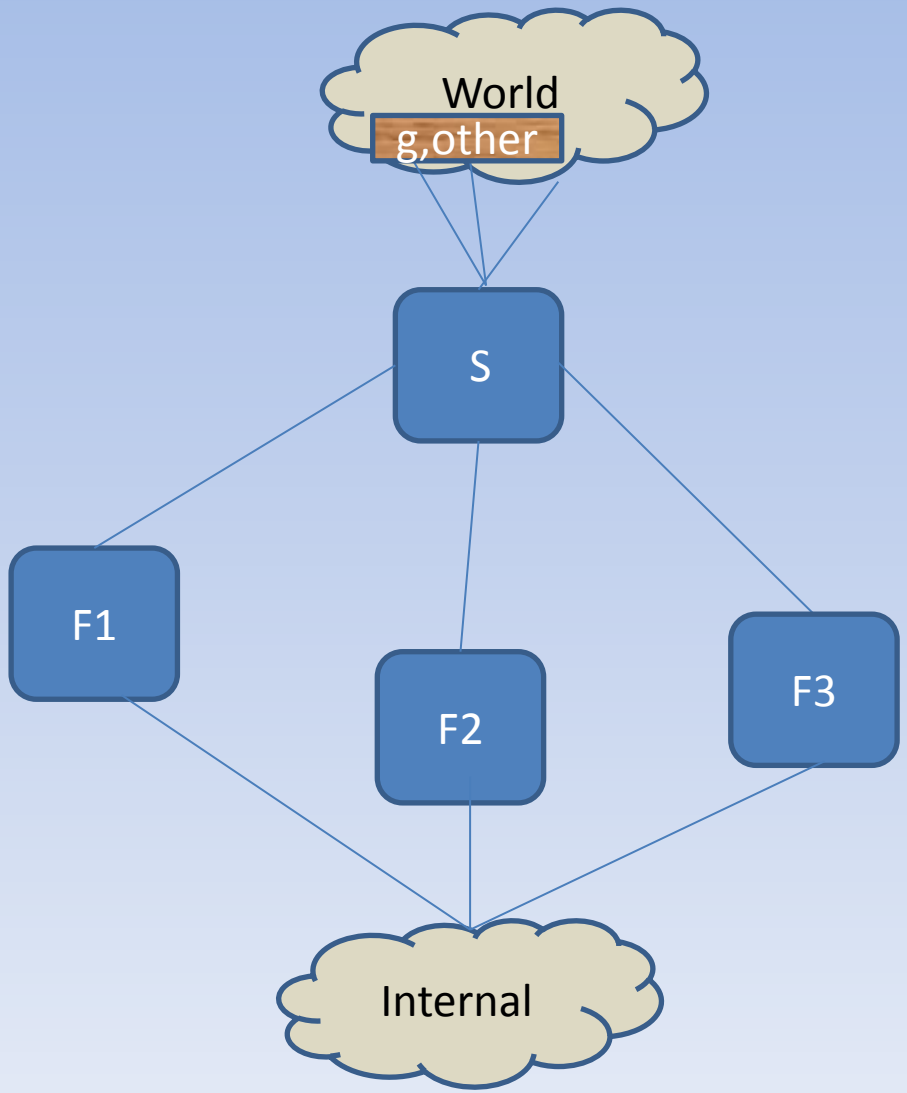
Example: firewall



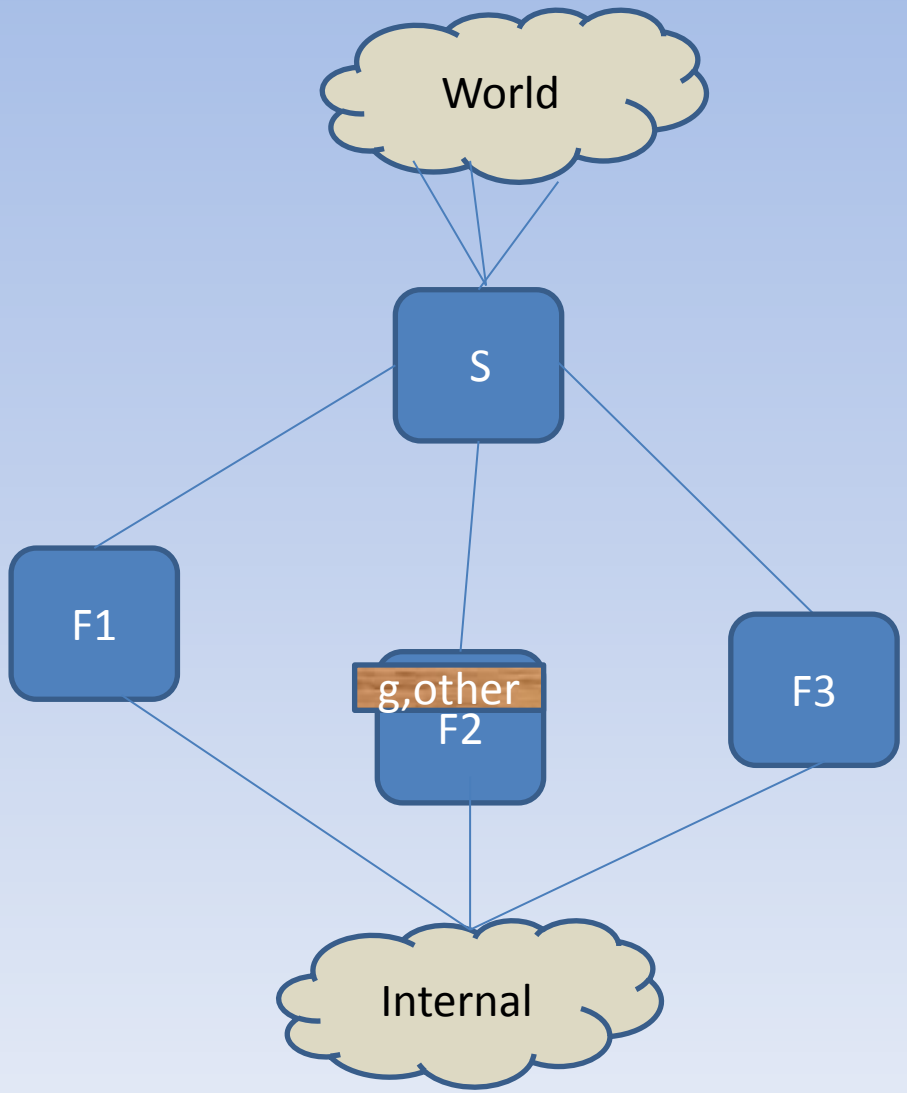
```
MODULE main
VAR
port : {S_0, F1_0, F2_0, F3_0, START, WORLD, DROP};
src : {Auth, Guest};
purpose : {Web, Other};
ASSIGN
next(port) := case
port = START : S_0;
port = S_0 & src = Auth : {F1_0, F2_0};
port = S_0 & src = Guest : F3_0;
port = F1_0 : WORLD;
port = F2_0 : WORLD;
port = F3_0 & purpose = Web : WORLD;
port = F3_0 & purpose = Other : DROP;
port = WORLD : WORLD;
port = DROP : DROP;
esac;
INIT port = START;

LTLSPEC  (purpose = Other & src = Guest -> port = DROP) &
((src = Auth | src = Guest & purpose = Web) -> F port = WORLD);
```

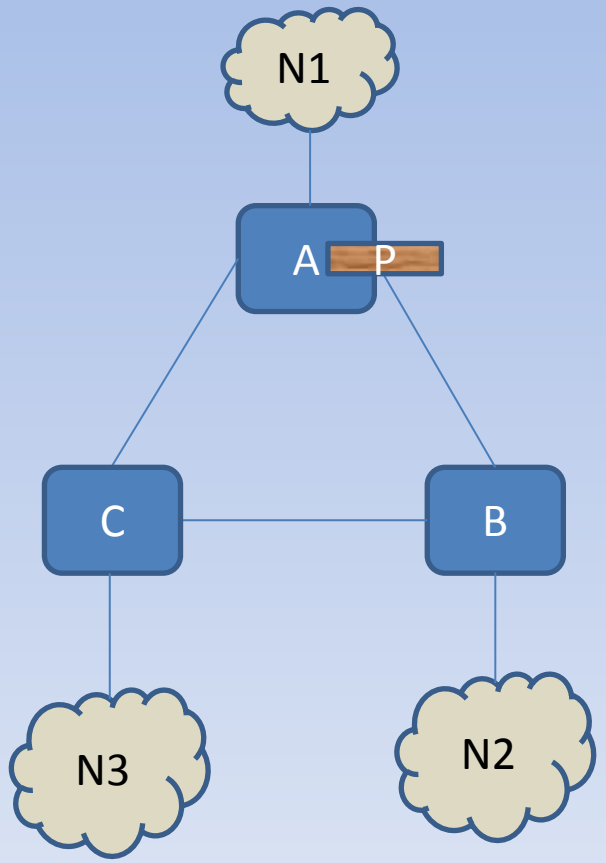
Example: firewall



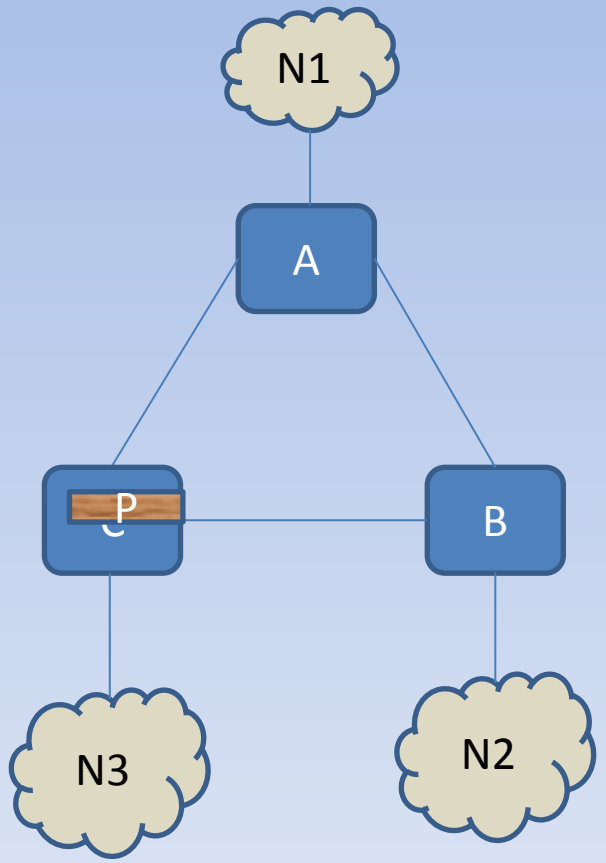
Example: firewall



Example: Cycle



Example: Cycle



A New Idea

- We'll use synthesis to generate update mechanisms automatically.
- Input:
  - Current configuration
  - target configuration
  - invariants.

- Output: Sequence of modifications to the forwarding rules of individual switches

Network Model

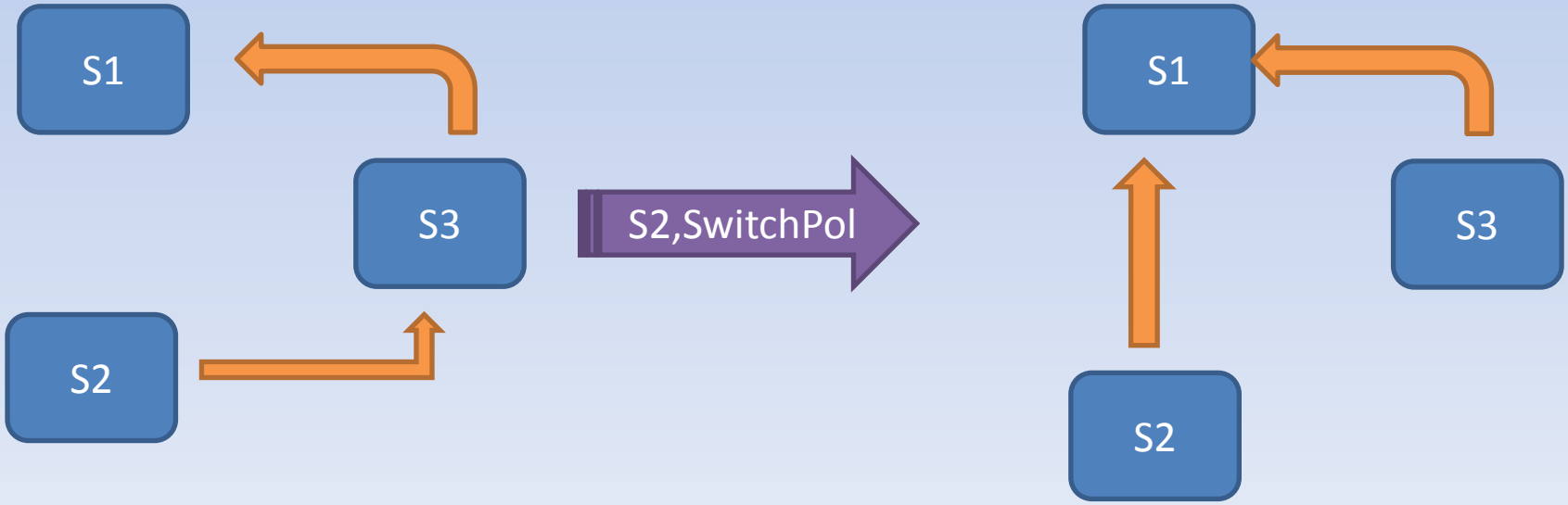
- *Network topology* = (Sw,P,inport,outport,ingress)
- Sw = Switches
- P = Ports
- $inport \in P \times Sw$
  for $p \in P$ there is a unique $s \in Sw$ with inport(p,s).
- $outport \in Sw \times P$
  for $p \in P$ there is a unique $s \in Sw$ with outport(s,p).
- Ingress = ports that get packets from WORLD.

Policies

- A *switch policy* *(switch's forwarding rules) is a policy of switch s if:*
  $SwitchPol$(p,pt) = (p',pt') s.t.
  - pt is a packet
  - (p,s) ∈ inport
  - (s,p') ∈ outport
- A *netwprk policy* *(network's forwarding rules)* is a function
  NetPol:S →SwitchPol s.t
  NetPol(s) = SwitchPol ⇒ SwitchPol is the switch policy of s.

Updates

- An *update* is a pair (s,SwitchPol) s.t. SwitchPol is a policy of s.
- $NetPol(s \leftarrow SwitchPol) \Rightarrow NetPol'(s) = SwitchPol$ and for all $s' \neq s$, NetPol'(s')=NetPol(s').

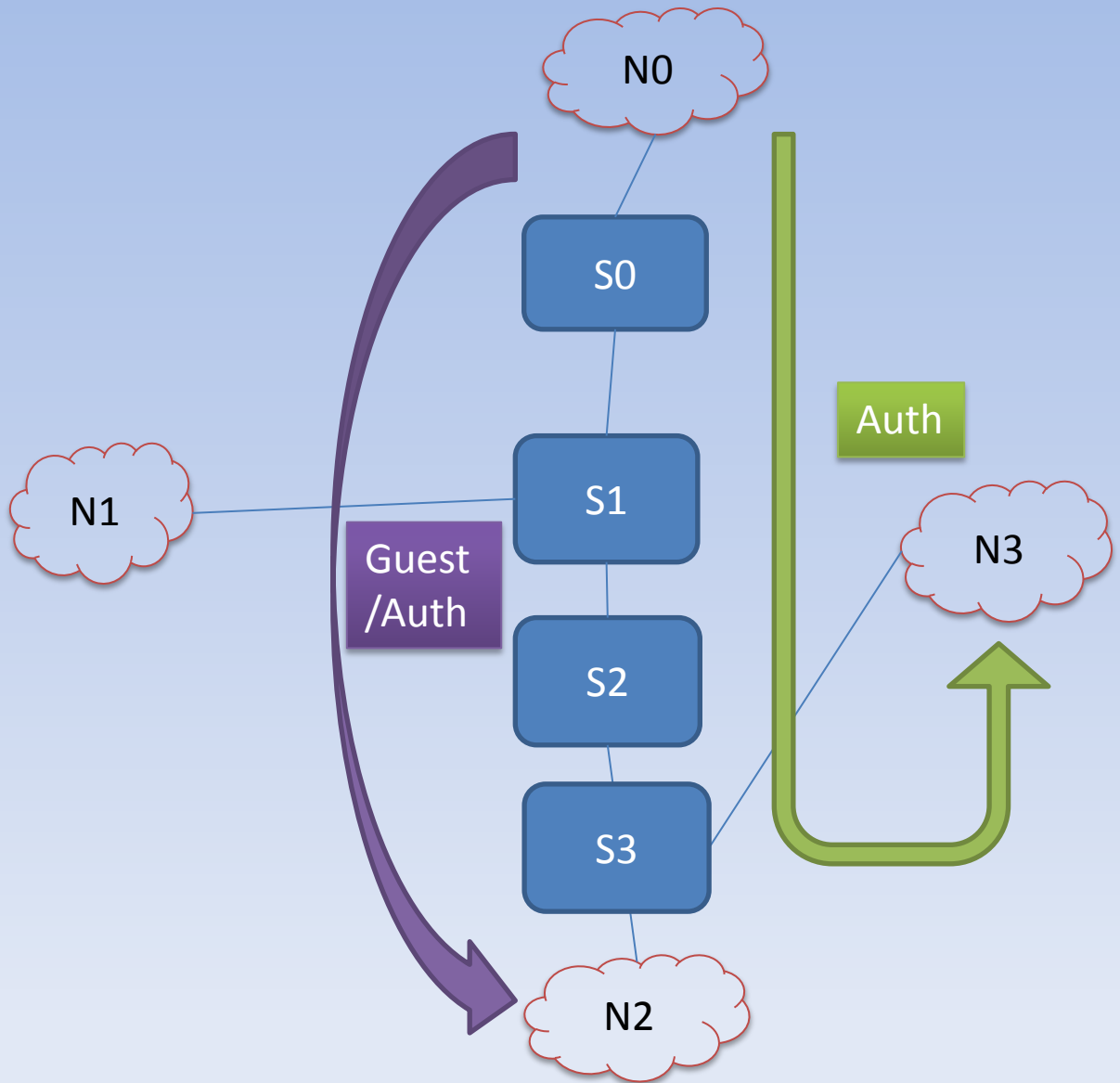


Commands

- A *command* is either an update or a wait command.

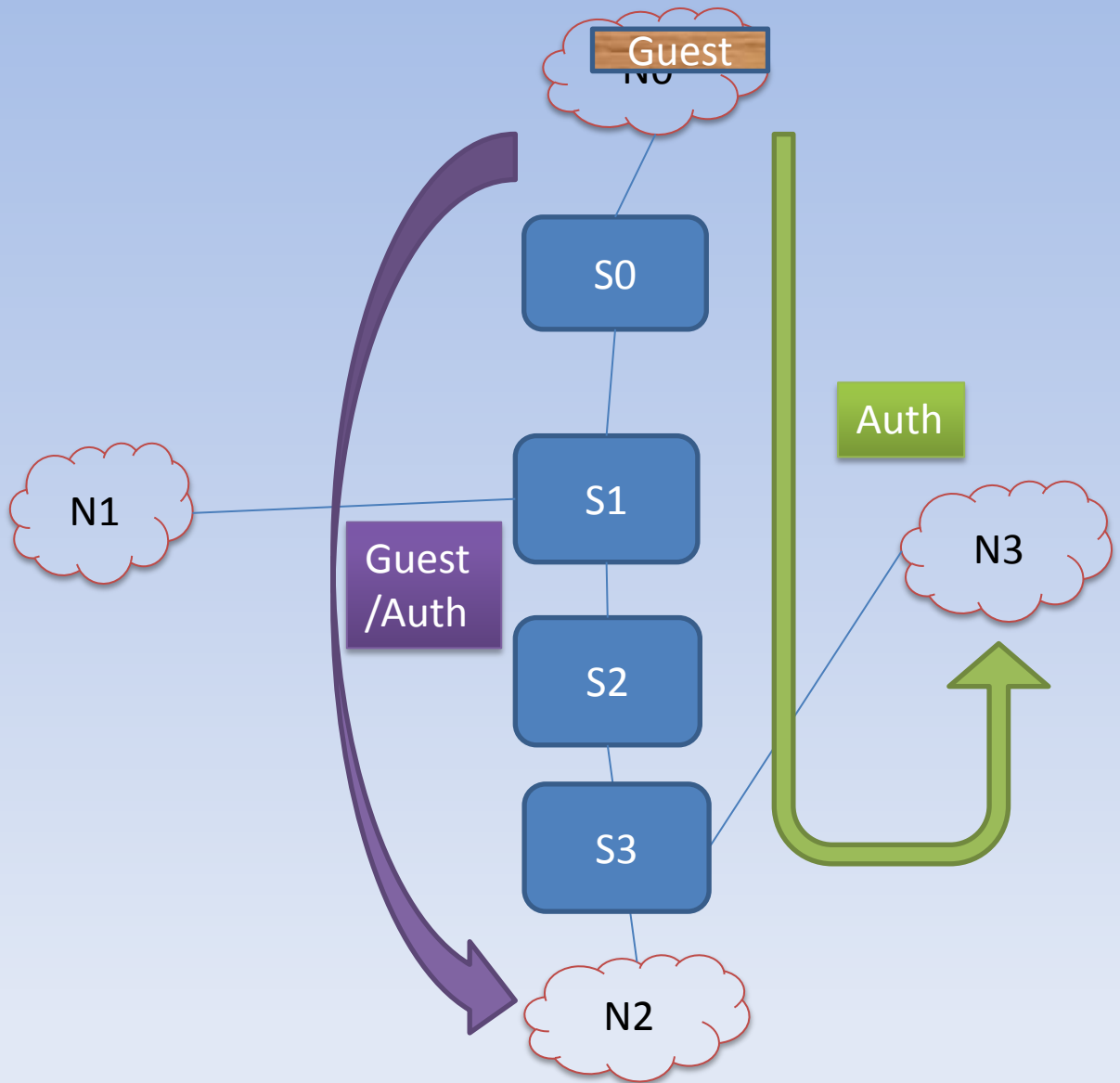
- A ***wait*** command:
  - Disables the ability to update network policy
  - Limits the network to one update while there'e a packet in it.

Why Wait is Important?



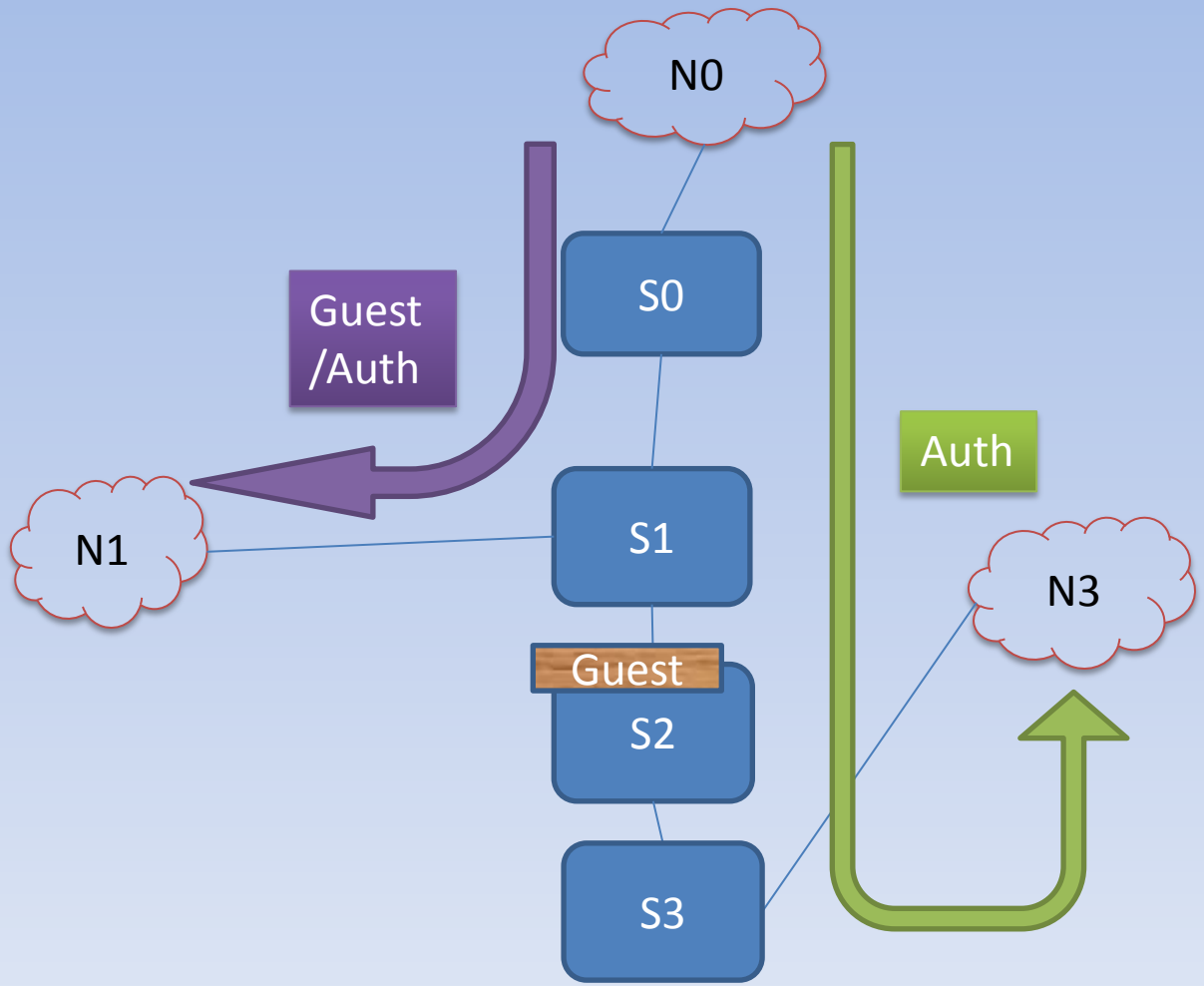
S3 checks whether it's Auth or guest before sending to N3

Why Wait is Important?



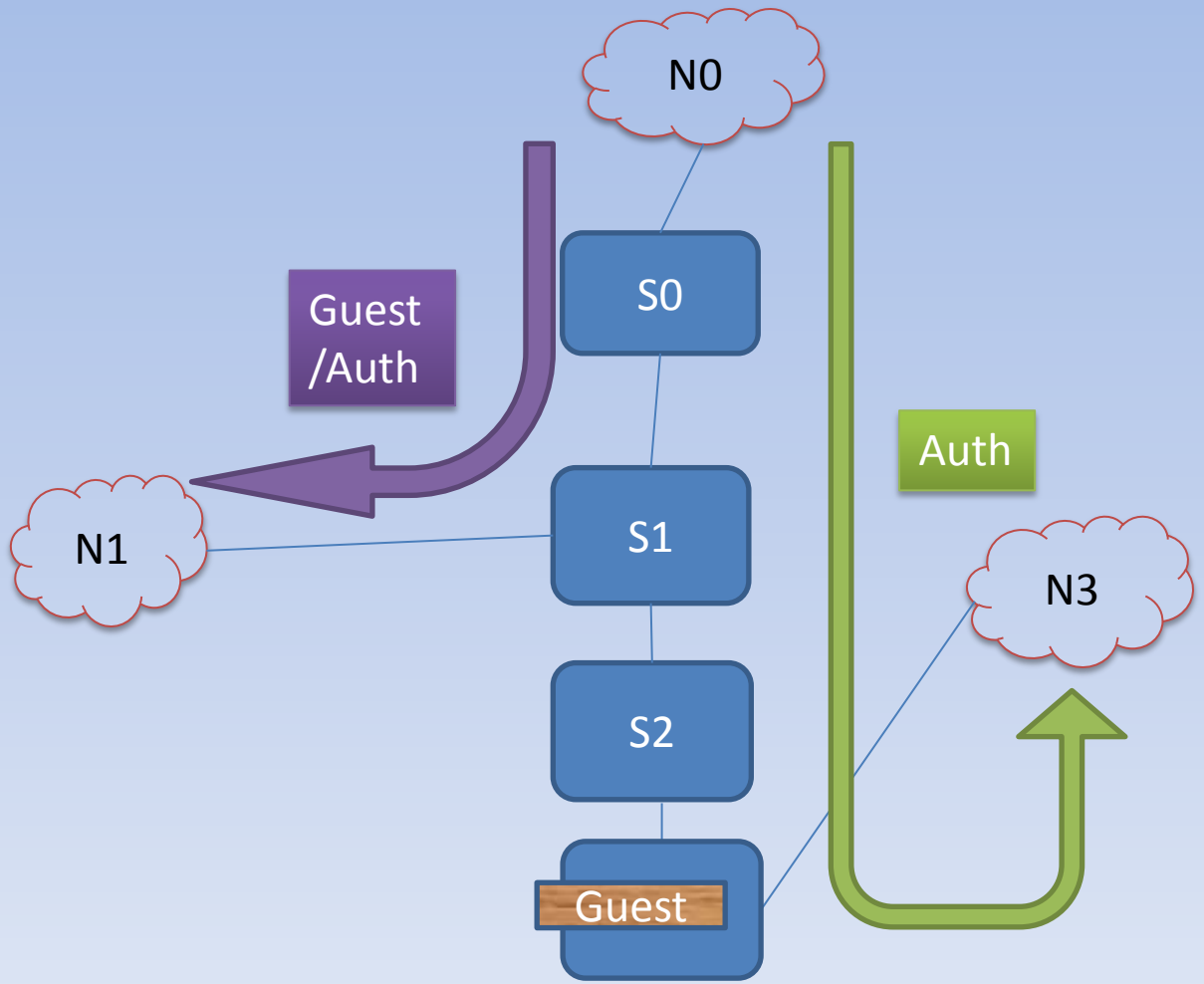
S3 checks whether it's Auth or guest before sending to N3

Why Wait is Important?



S3 checks whether it's Auth or guest before sending to N3

Why Wait is Important?



S3 does not check Packet's src before sending to N3

Network States & Transition

- A *network state* ns = ((p,pt),NetPol,wait-flag,comSeq)
- A *network transition* is a relation $ns \rightarrow ns'$.
- There are 4 types of transitions:
  1. A ***packet move***:
     ns = ((p,pt),NetPol,wait-flag,comSeq)
     ns' = ((p',pt'),NetPol,wait-flag,comSeq)
  2. An ***update transition:***
     ns = ((p,pt),NetPol, false, (s, SwitchPol).comSeq)
     ns' = ((p,pt),,NetPol[$s \leftarrow SwitchPol$],false,comSeq)
  3. A ***wait transition:***
     ns = ((p,pt),NetPol,wait-flag,wait.comSeq)
     ns' = ((p,pt),NetPol,true,comSeq)

Network Transition

4. A ***<u>new packet transition:</u>***
   ns = ((p,pt),NetPol,wait-flag,comSeq)
   ns'= ((p',pt'),NetPol,false,comSeq)
   where $p' \in ingress$.

Network Trace

- A *network trace* nt is an infinite sequence $ns_0 ns_1 ns_2 \ldots$ s.t. for all i ≥ 0 $ns_i \rightarrow ns_{i+1}$ is a network transition.

- A network state ns is wait-correct if there are no loops in its network policy.

Update Synthesis Problem

- Given:
  - an initial network policy $NetPol_i$,
  - a final network policy $NetPol_f$,
  - specification φ,

  construct a sequence of commands comSeq such that:
  - $NetPol_i \rightarrow NetPol_f$ *and*
  - comSeq is correct with respect to φ and $NetPol_i$.

Procedure ORDERUPDATE($NetPol_i$, $NetPol_f$, φ)

Input: Initial network policy $NetPol_i$, final network policy $NetPol_f$, and LTL specification φ.

Output: Simple and careful sequence of switch updates L, if it exists

```
if hasLoops(NetPol_i) ∨ hasLoops(NetPol_f) then
    return "Loops in initial or final configuration."
else
    W ← false                                   ▷ Wrong configurations.
    V ← false                                   ▷ Visited configurations.
    (ok, L) ← DFSforOrder(NetPol_i, ⊥)
    if ok then
        return L
    else
        return "No simple and careful update sequence exists."
```

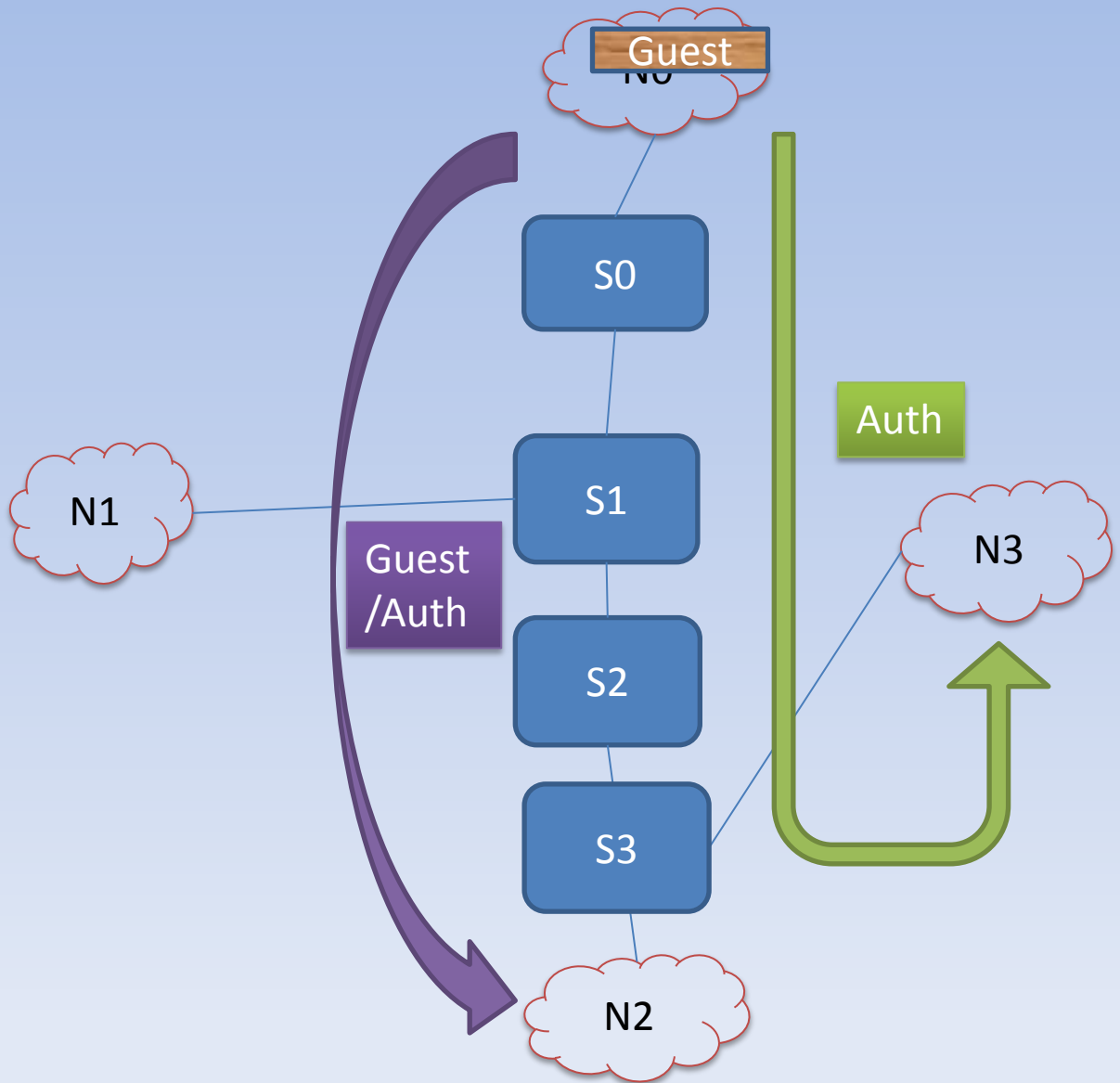
Procedure DFSFORORDER($NetPol, cs$)

Input: Current network policy $NetPol$, most recently updated switch cs.

Output: Boolean ok if a correct update sequence exists; L correct sequence of switch updates

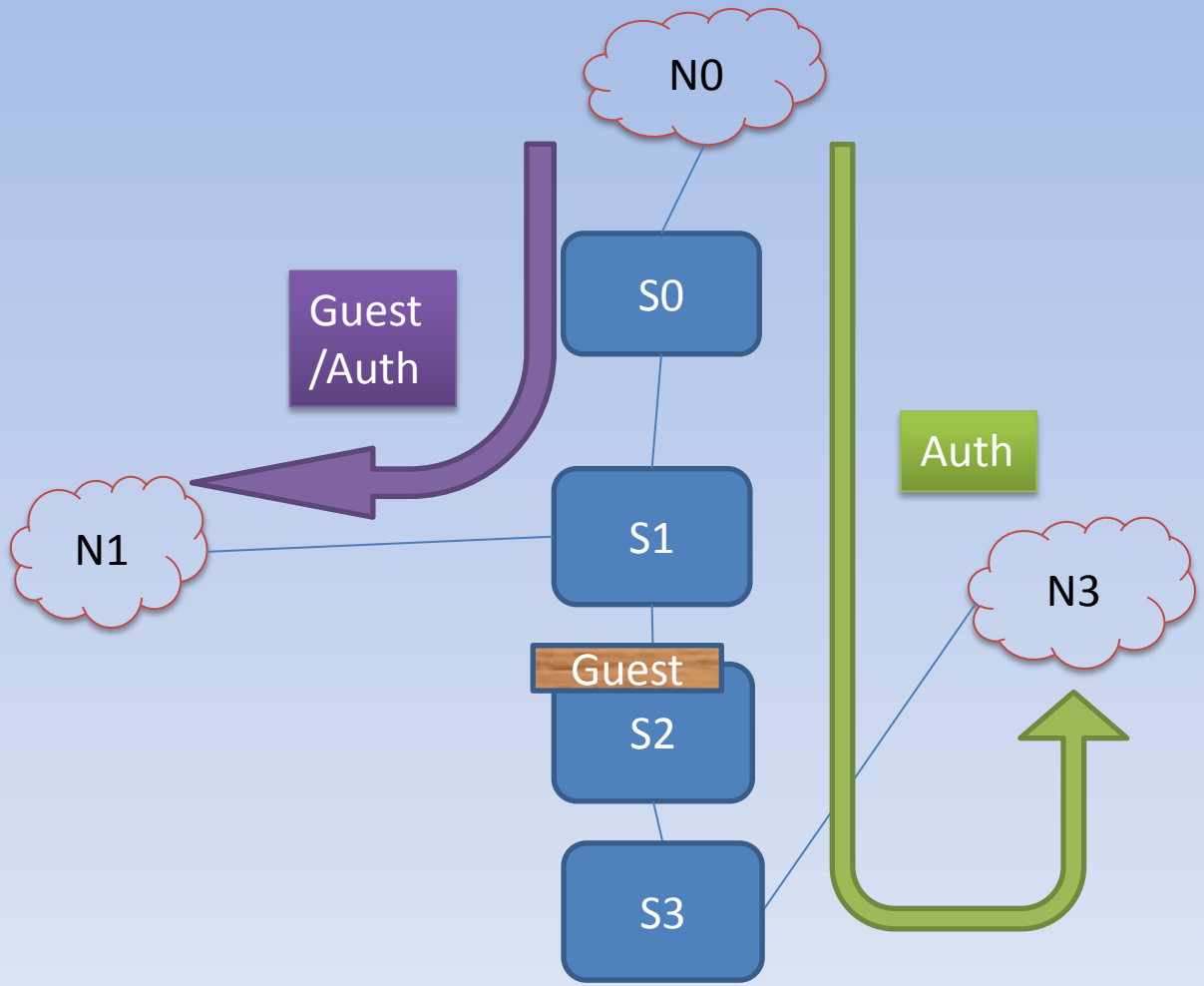
```
if NetPol = NetPol_f then
    return (true,[NetPol])                          ▷ Reached final configuration.
if NetPol ⊨ V then
    return (false,[])                               ▷ Already visited NetPol.
V ← V ∨ NetPol                                      ▷ Add to visited configurations.
if NetPol ⊨ W then
    return (false,[])                               ▷ Previous counterexample applies.
if cs ≠ ⊥ then                                      ▷ If there was a previous update,
    (ok,cex) ← hasNewLoops(NetPol,cs)               ▷ Check for forwarding loops.
    if (¬ ok) then
        W ← W∨ analyzeCex(cex)                      ▷ Learn from loop counterexample.
        return (false,[])
(ok,cex) ← ModelCheck(NetPol,φ)
if (¬ ok) then
    W ← W∨ analyzeCex(cex)                          ▷ Learn from property counterexample.
    return (false,[])
for all (NetPol_next, cs) ∈ NextPolicies(NetPol) do ▷ Try to update one more switch.
    (ok,L) ← DFSforOrder(NetPol_next, cs)           ▷ Recursive call.
    if ok then
        return (true,NetPol :: wait :: L)
return (false,[])
```

Running Example



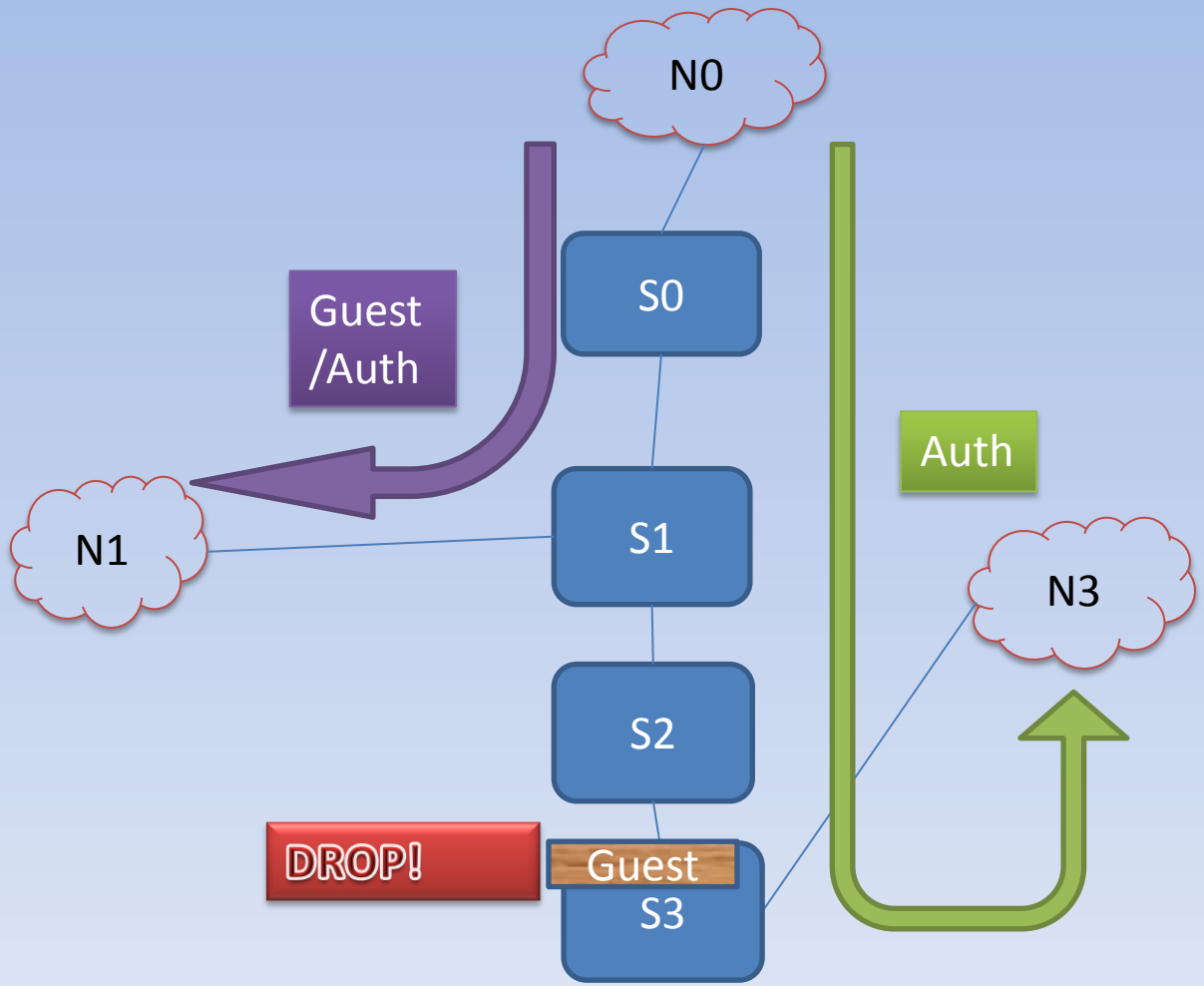
S3 checks whether it's Auth or guest before sending to N3

Running Example



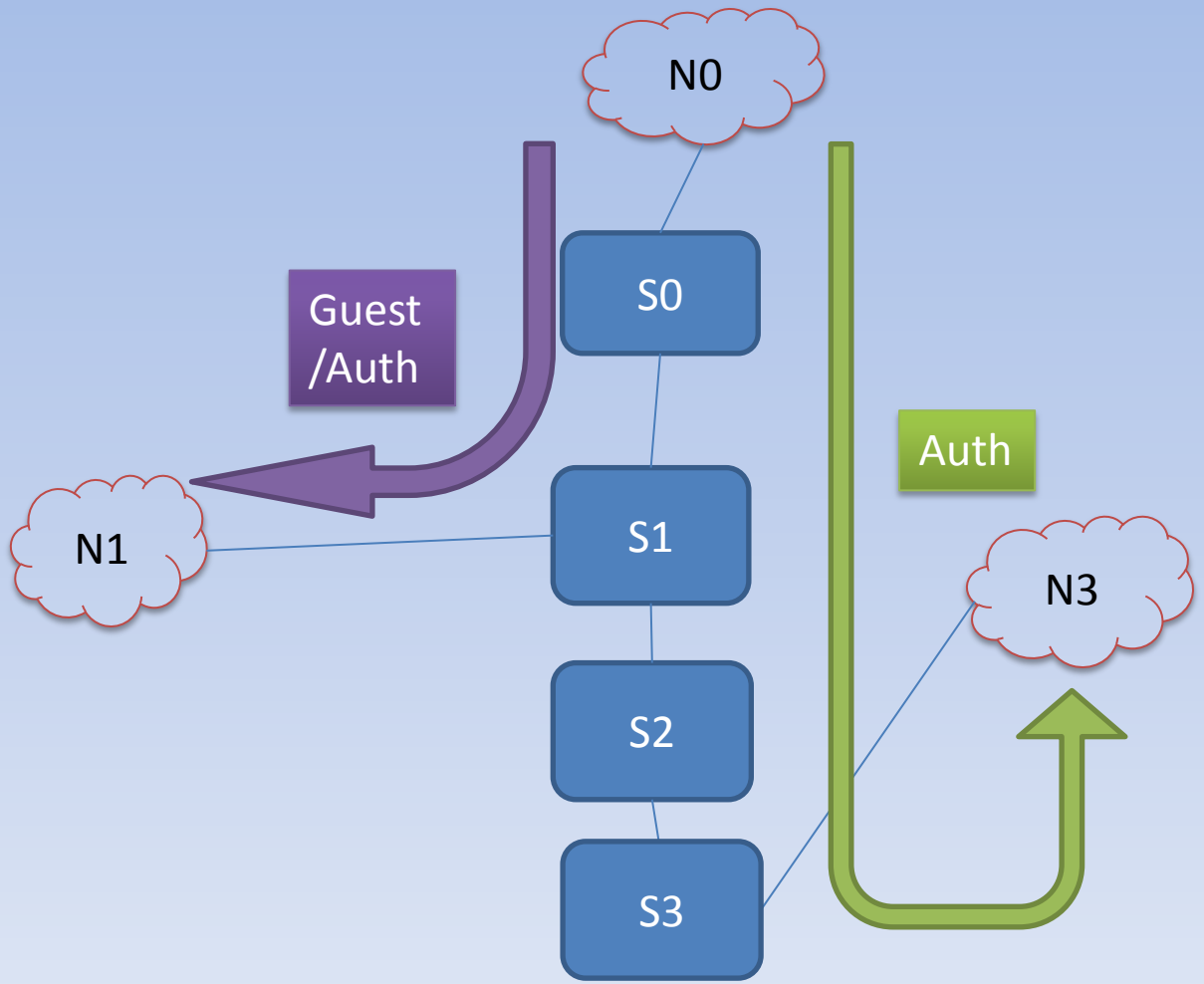
S3 checks whether it's Auth or guest before sending to N3

Running Example



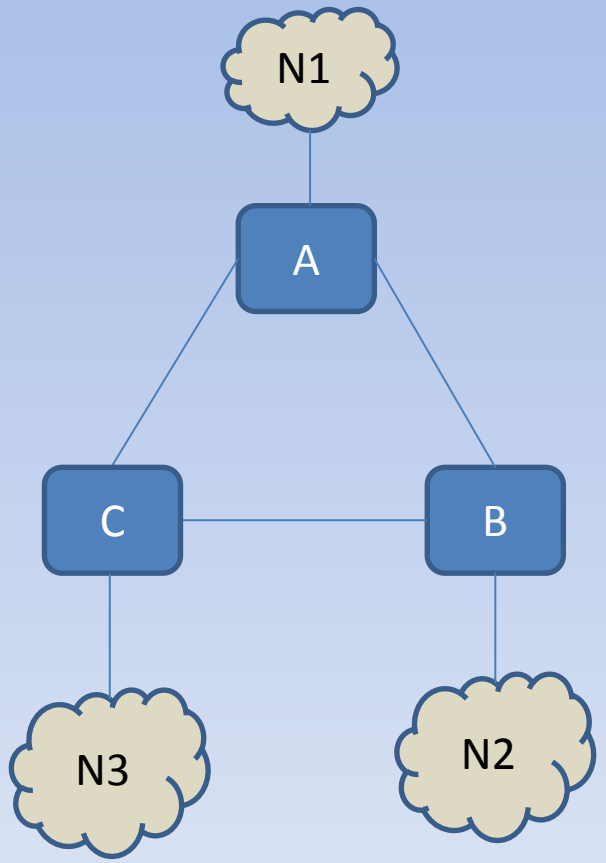
S3 checks whether it's Auth or guest before sending to N3

Running Example

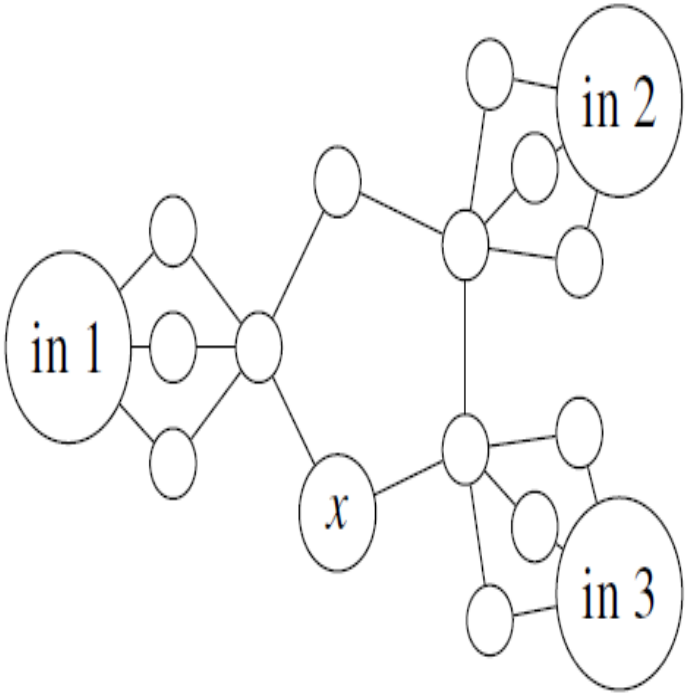


S3 does not check Packet's src before sending to N3

Example: Cycle



Experiments



Time of updating with varying number of switches in network

Algorithm	100 Nodes	250 Nodes	500 Nodes	1000 Nodes
OrderUpdate	10	83	355	1101

Time of updating with varying number of switches to update

Nodes	5	6	7	8	9	10	11	12	13	14	15	30	60
Time	165	142	166	222	222	205	273	276	354	339	370	611	2106

Impossible updates

Algorithm	100 Nodes	250 Nodes	500 Nodes	1000 Nodes
OrderUpdate	19	170	900	3963
OrderUpdate w/o counterexamples	101	1793	6269	Timeout

Summary

- The running times are too large for online use.
- The purpose was to build a prototype tool to confirm the feasibility of the new approach.
- The counterexample analysis has great influence on the running times, though.
- If there is no way of safe update it takes much longer to finish running, because there is a large number of update sequences possible

The End