# SPAIN –
# Smart Path Assignment In Networks

**Radosław Pudełkiewicz**

Uniwersytet Warszawski

Wydział Matematyki, Informatyki i Mechaniki

19 stycznia 2011

# SPAIN in single phase

SPAIN provides multipath forwarding using inexpensive, commodity off-the-shelf (COTS) Ethernet switches, over arbitrary topologies.

# Why Ethernet?

Ethernet is the primary network technology for data centers:

- Ubiquity
- self-configuration
- high link bandwidth at low cost

# Why Ethernet is hard to scale?

Ethernet's lack of scalability stems from three main problems:

- Its use of the Spanning Tree Protocol to automatically ensure a loop-free topology.
- Packet floods for learning host locations.
- Host-generated broadcasts, especially for ARP.

# Currently used  solution.

„Adding IP (Layer-3) routers “solves” the scaling problem via the use of subnets, but introduces new problems, especially the difficulty of supporting dynamic mobility of virtual machines”

# Existing proposals for improving Ethernet scalability

Table 1: Comparing SPAIN against related work

| | SPAIN | SEATTLE [22] | TRILL [7] | VL2 [17] | PortLand [27] | MOOSE [31] |
|---|---|---|---|---|---|---|
| Wiring Topology | Arbitrary | Arbitrary | Arbitrary | Fat-tree | Fat-tree | Arbitrary |
| Usable paths | Arb. multiple paths | Single Path | ECMP | ECMP | ECMP | Single Path |
| Deploy incrementally? | YES | NO | YES | NO | NO | YES |
| Uses COTS switches? | YES (L2) | NO | NO | YES (L3) | NO | NO |
| Needs end-host mods? | YES | NO | NO | YES | NO | NO |

Fat-tree = multi-rooted tree; ECMP = Equal Cost Multi-Path.

# The design of SPAIN

SPAIN goals are to:

- Deliver more bandwidth and better reliability than spanning tree.
- Support arbitrary topologies, not just fat-tree or hypercube, and extract the best bisection bandwidth from any topology.
- Utilize unmodified, off-the-shelf, commoditypriced (COTS) Ethernet switches.
- Minimize end host software changes, and be incrementally deployable.

# Offline configuration of the network in SPAIN

These algorithms address several challenges:

- Which set of paths to use?
- How to map paths to VLANs?
- How to handle unplanned topology changes?

# Path-set computation

**Algorithm 1** Algorithm for Path Computation

Given:
  $G_{full} = (V_{full}, E_{full})$: The full topology,
  $w$: Edge weights,
  $s$: Source, $d$: Destination
  $k$: Desired number of paths per $s, d$ pair

Initialize: $\forall e \in E : w(e) = 1$
/* `shortest` computes weighted shortest path */
Path $p$ = `shortest`$(G, s, d, w)$ ;
**for** $e \in p$ **do**
  $w(e) += |E|$

**while** $(|P| < k)$ **do**
  $p$ = `shortest`$(G, s, d, w)$
  **if** $p \in P$ **then**
    /* no more useful paths */
    *break ;*
  $P = P \bigcup \{p\}$
  **for** $e \in p$ **do**
    $w(e) += |E|$

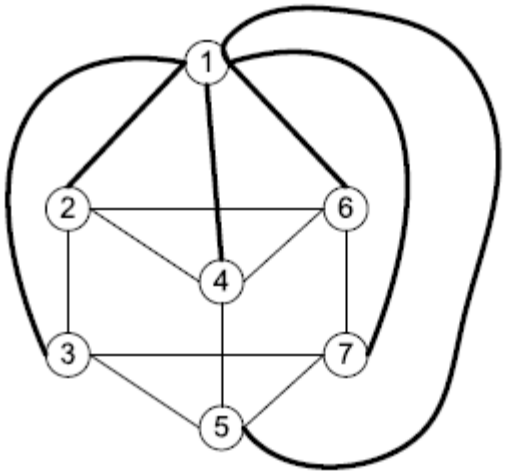
return $P$

# Mapping path sets to VLANs

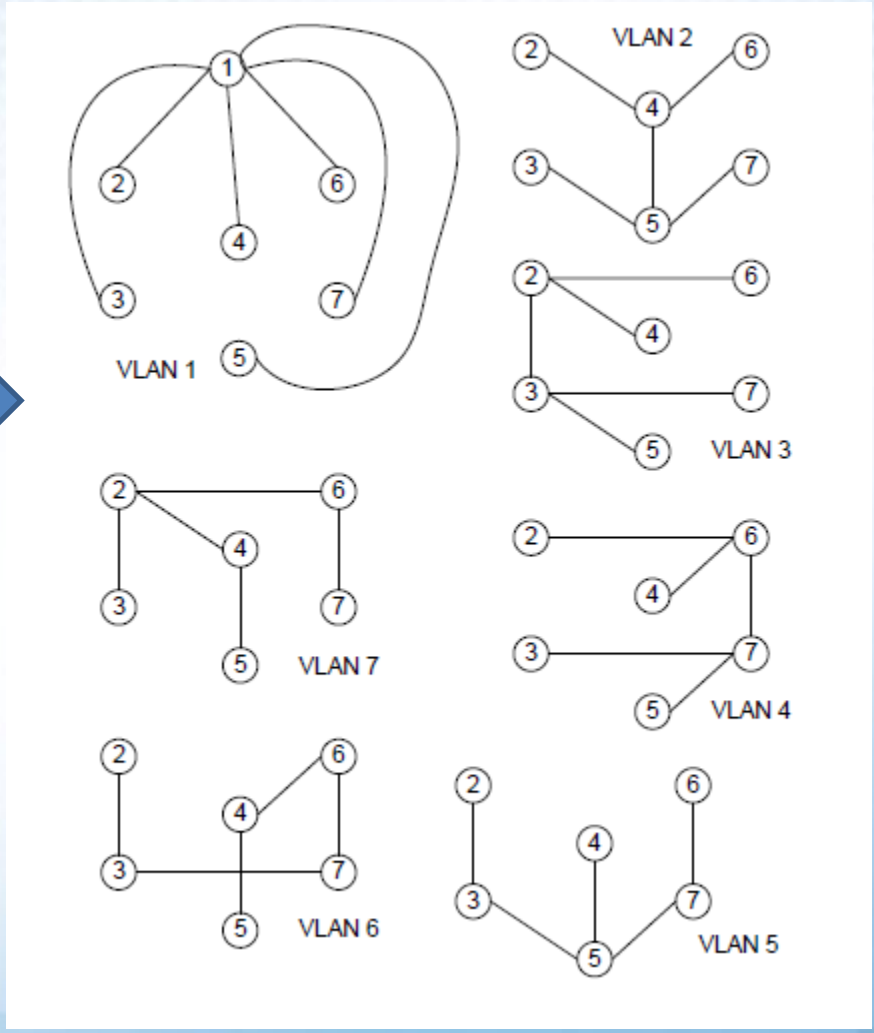
**Algorithm 2** Greedy VLAN Packing Heuristic

```
1:  Given: G = (V, E), k
2:  SG = ∅ /* set of loop-free subgraphs*/
3:  for v ∈ V do
4:      for u ∈ V do
5:          P =ComputePaths(G, v, u, k) ;
6:          for p ∈ P /* in a random order */ do
7:              if p not covered by any graph in SG then
8:                  Success = FALSE;
9:                  for S ∈ SG /* in a random order */ do
10:                     if p does not create loop in S then
11:                         Add p to S
12:                         Success = TRUE ;
13:                 if Success == FALSE then
14:                     S' = new graph with p
15:                     SG = SG ⋃ {S'}
16: return SG
```

# En example




7-switch topology; original tree in bold



# End-host algorithms

An end host uses the following data structures and parameters:

- ES(m): the ID of the edge switch to which MAC address m is currently connected.
- Vreach(es): the set of VLANs that reach the edge switches.
- R: the reachability VLAN map, a bit map encoding the union of Vreach(•) over all es.
- Vusable(es): the set of VLANs that have recently tested as usable to reach es.
- Trepin is the length of time after which non-TCP flows go through the VLAN re-pinning process.
- Tsent is the minimum amount of time since last send on a VLAN that triggers a chirp (see below).
- Vsent(es): the set of VLANs that we sent a packet via es within the last Tsent seconds.

# Sending a Packet- Selecting a VLAN

**Algorithm 3** Selecting a VLAN

```
1:  /* determine the edge switch of the destination */
2:  m = get_dest_mac(flow)
3:  es = get_es(m)
4:  /* candidate VLANs: those that reach es */
5:  if candidate_vlans is empty then
6:     /* No candidate VLANs; */
7:     /* Either es is on a different SPAIN cloud or m is a non-
       SPAIN host */
8:     return the default VLAN (VLAN 1)
9:  /* see if any of the candidates are usable */
10: usable_vlans = candidate_vlans ⋂ V_usable(es)
11: if usable_vlans is empty then
12:    return the default VLAN (VLAN 1)
13: init_probe(candidate_vlans − usable_vlans)
14: return a random v ∈ usable_vlans.
```

# Sending a Packet – Periodic VLAN re-selection

**Algorithm 4** Determine if a flow needs VLAN selection

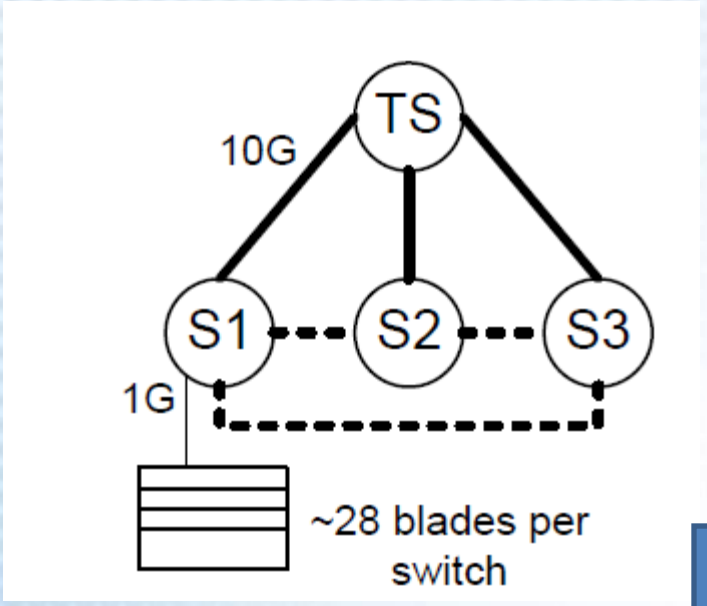
```
1:  if last_move_time >= last_pin_time then
2:     /* we moved since last VLAN selection - re-pin flow */
3:     return true;
4:  current_es = get_es(dst_mac)
5:  if saved_es (from the flow state) ! = current_es then
6:     /* destination moved – update flow state & re-pin */
7:     saved_es = current_es;
8:     return true
9:  if current_vlan(flow) ≤ 0 then
10:    return true /* new flows need VLAN selection */
11: if proto_of(flow) ≠ TCP then
12:    if (now − last_pin_time) ≥ T_repin then
13:       return true /* periodic re-pin */
14: else
15:    if cwnd(flow) ≤ W_repin_thresh && is_rxmt(flow)
       then
16:       return true /* TCP flow might prefer another path */
17: return false /* no need to repin */
```

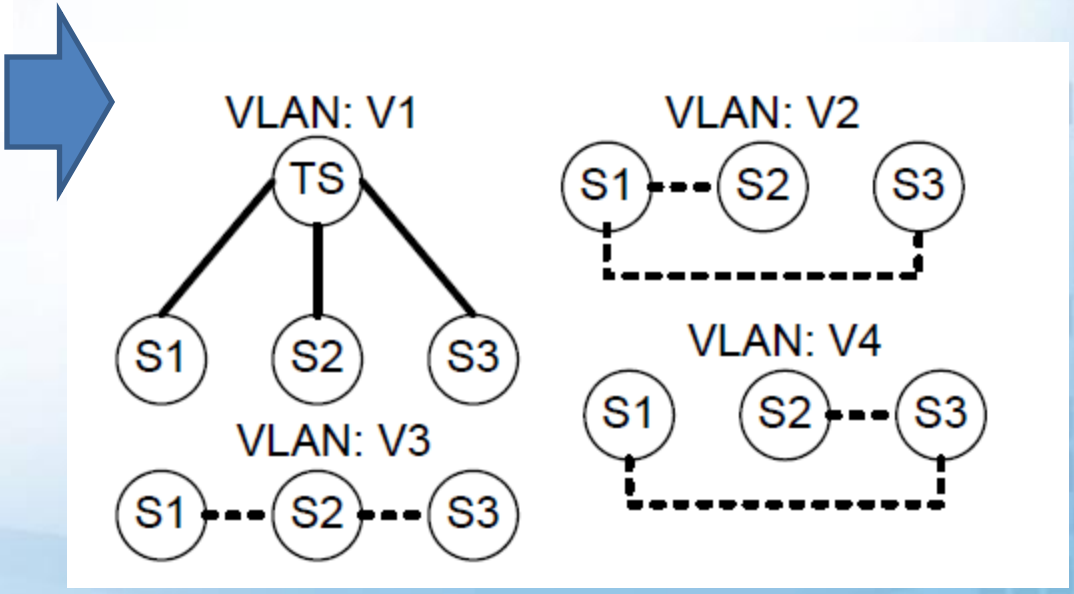
# Reciving a Packets

**Algorithm 5** Receiving a Packet

```
1:  vlan = get_vlan(packet)
2:  m = get_src_mac(packet)
3:  if is_chirp(packet) then
4:     update_ARP_table(packet)
5:     update_ES_table(packet, vlan)
6:     if wants_chirp_response(packet) then
7:        send_unicast_chirp(m, vlan)
8:  es = get_es(m) /* determine sender's edge switch */
9:  /* mark packet-arrival VLAN as usable for es */
10: V_usable(es) = V_usable(es) ⋃ vlan
11: /* chirp if we haven't sent to es via vlan recently */
12: if the vlan bit in V_sent(es) is not set then
13:    send_unicast_chirp(m, vlan)
14:    /* V_sent(es) is cleared every T_sent sec. */
15: deliver packet to protocol stack
```

# Experimental evaluation



# End-host overheads

| Configuration | ping RTT (usec) | TCP throughput (Mbit/sec) | |
|---|---|---|---|
| | | 1-way | 2-way, [min,max] |
| Unmodified Linux | 98 | 990 | 1866 [1858,1872] |
| 1st pkt, cold start | 4.03 ms | | |
| SPAIN, no chirping | 98 | 988 | 1860 [1852,1871] |
| 1st pkt, cold start | 3.94 ms | | |
| SPAIN w/chirping | 99 | 988 | 1866 [1857,1876] |
| 1st pkt, cold start | 4.84 ms | | |

ping results: mean of 100 warm-start trials;
throughput: mean, min, max of 50 trials

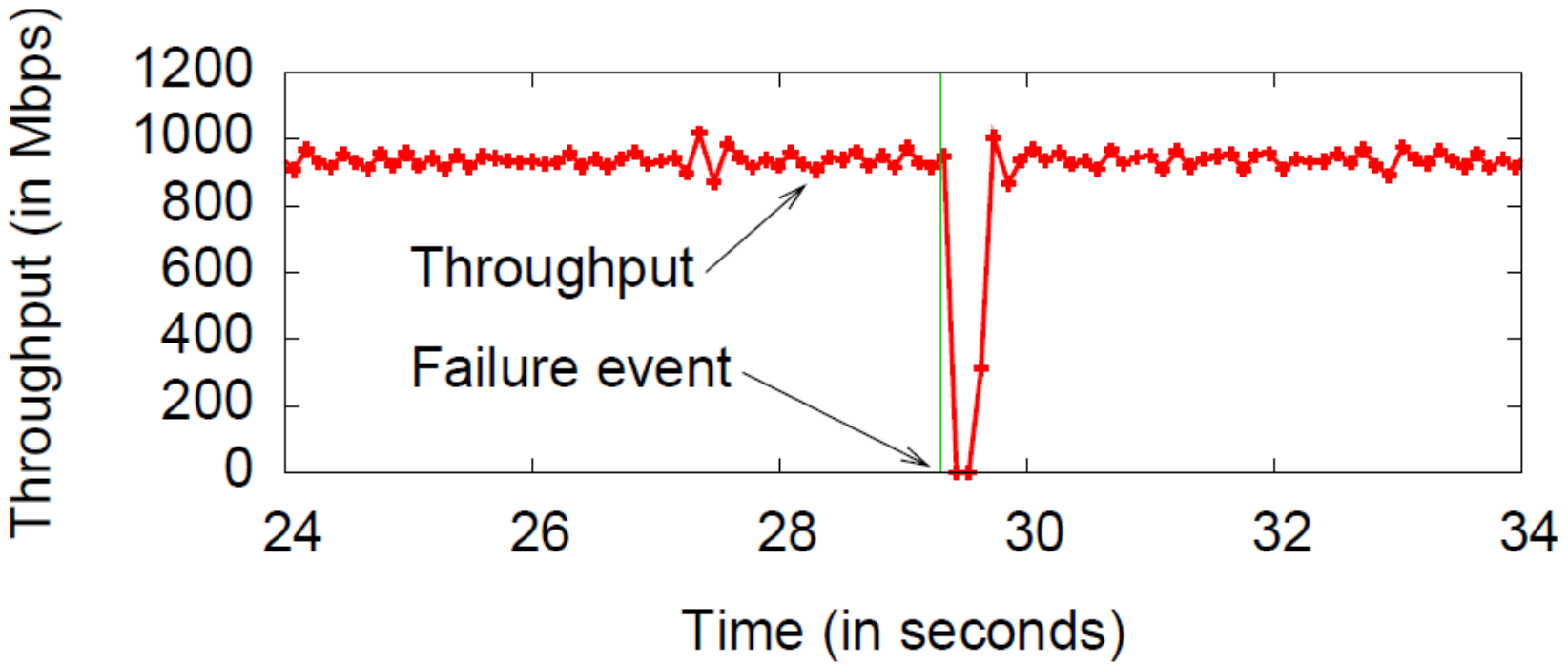
# SPAIN vs. spanning tree

Table 5: Spanning-tree vs. SPAIN

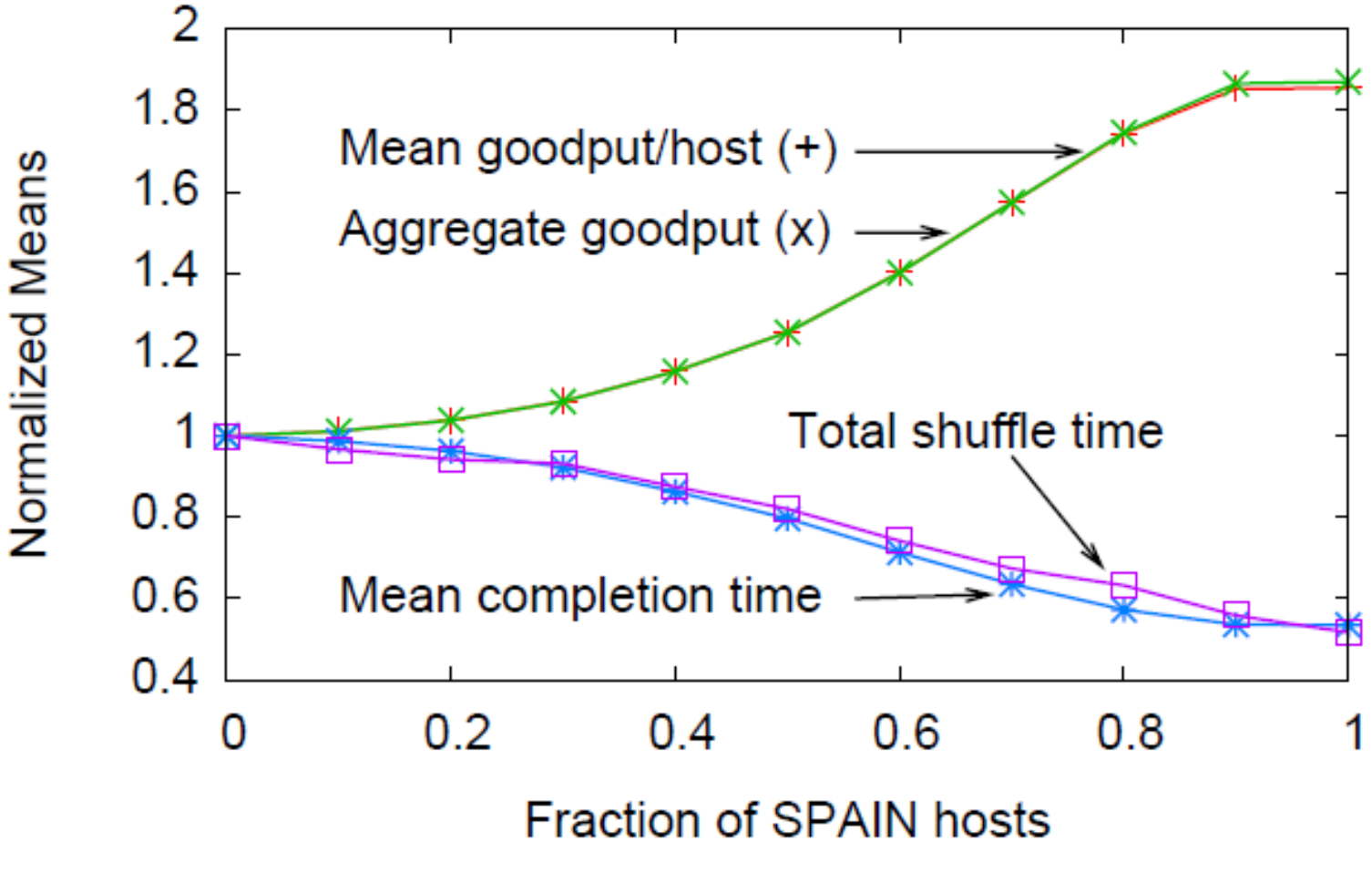
| | Spanning Tree | SPAIN |
|---|---|---|
| Mean goodput/host (Mb/s) | 449.25 | 834.51 |
| Aggregate goodput (Gb/s) | 35.60 | 66.68 |
| Mean completion time/host | 744.57 s | 397.50 s |
| Total shuffle time | 831.95 s | 431.12 s |

Results are means of 10 trials, 500 MBytes/trial, 80 hosts

# Fault tolerance



# Incremental deployability




Results are means over 10 trials

# Summary and conclusions

„SPAIN improves aggregate goodput over spanning-tree by 87% on a testbed (…).”