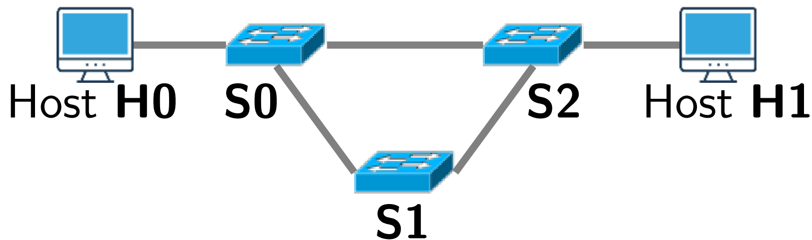
Bayonet: Probabilistic Inference for Networks

Timon Gehr

Department of Computer Science
ETH Zürich

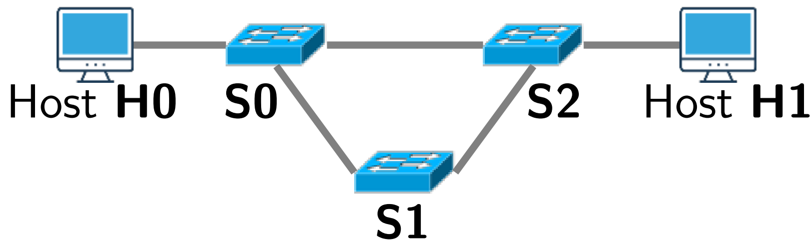
March 12, 2018

Introduction



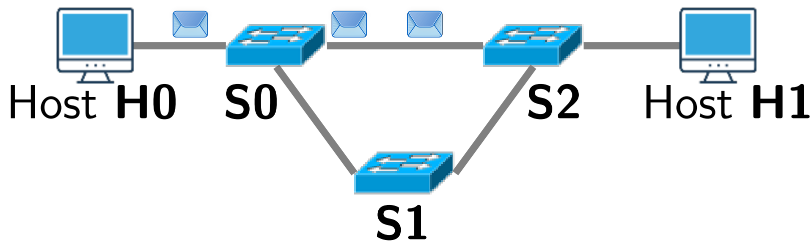
Want to understand behavior of computer network.
Issue: Uncertainty.

Introduction



Want to understand behavior of computer network.
Issue: Uncertainty. Sources:

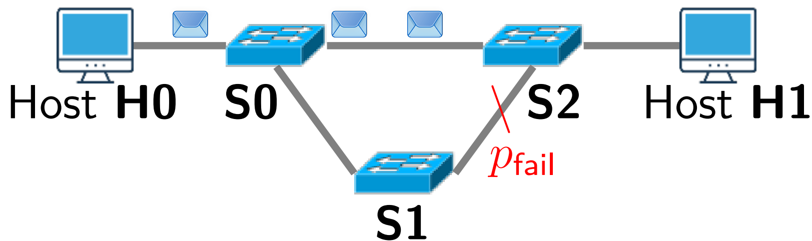
Introduction



Want to understand behavior of computer network.
Issue: Uncertainty. Sources:

- Traffic

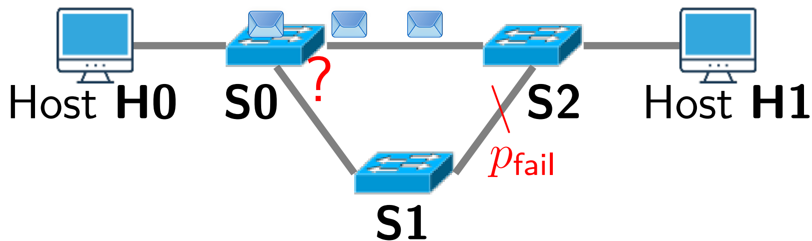
Introduction



Want to understand behavior of computer network.
Issue: Uncertainty. Sources:

- Traffic
- Failures

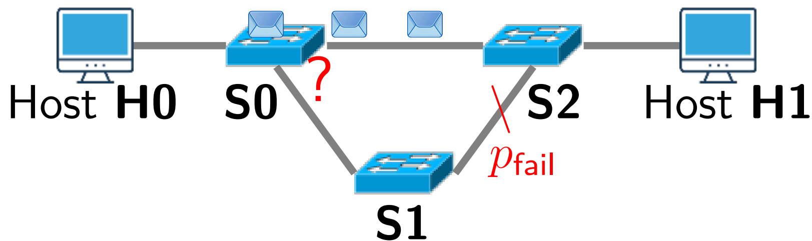
Introduction



Want to understand behavior of computer network.
Issue: Uncertainty. Sources:

- Traffic
- Failures
- Load-balancing (e.g ECMP).

Introduction



Want to understand behavior of computer network.
Issue: Uncertainty. Sources:

- Traffic
- Failures
- Load-balancing (e.g ECMP).
- ...

Simplified Computer Network Model

Networks

Network is given as a set of nodes V with bidirectional links L between *interfaces*. E.g. $\{(\mathsf{S0}, \mathsf{pt2}), (\mathsf{S2}, \mathsf{pt1})\} \in L$.

Simplified Computer Network Model

Networks

Network is given as a set of nodes V with bidirectional links L between *interfaces*. E.g. $\{(\mathsf{S0}, \mathsf{pt2}), (\mathsf{S2}, \mathsf{pt1})\} \in L$.

Packet Queues

Each node one input and one output queue. Queues may have a limited *capacity*.

Simplified Computer Network Model

Networks

Network is given as a set of nodes V with bidirectional links L between *interfaces*. E.g. $\{(\mathsf{S0}, \mathsf{pt2}), (\mathsf{S2}, \mathsf{pt1})\} \in L$.

Packet Queues

Each node one input and one output queue. Queues may have a limited *capacity*.

Node Programs

- Each node runs a (stateful) *Node Program*.

Simplified Computer Network Model

Networks

Network is given as a set of nodes V with bidirectional links L between *interfaces*. E.g. $\{(\mathsf{S0}, \mathsf{pt2}), (\mathsf{S2}, \mathsf{pt1})\} \in L$.

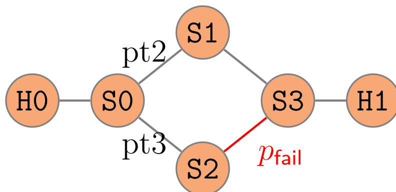
Packet Queues

Each node one input and one output queue. Queues may have a limited *capacity*.

Node Programs and Scheduling

- Each node runs a (stateful) *Node Program*. Execution is interleaved: In each step, either run a node program until externally-observable action, or forward a packet.
- Randomized scheduler selects next action.

Bayonet Program: Reliability




```
1  topology{
2    nodes{ H0, H1, S0, S1, S2, S3 }
3    links{
4      (H0,pt1) <-> (S0,pt1),
5      (S0,pt2) <-> (S1,pt1),
6      (S0,pt3) <-> (S2,pt1),
7      (S1,pt2) <-> (S3,pt1),
8      (S2,pt2) <-> (S3,pt2),
9      (S3,pt3) <-> (H1,pt1)
10   }
11 }
```

Bayonet Program: Reliability

```
1 def h0(pkt, port){
2   fwd(1);
3 }
4 def h1(pkt, port)
5   state arrived(0)
6 {
7   arrived=1;
8   drop;
9 }
10 def s0(pkt, port){
11   if flip(1/2){
12     fwd(2);
13   }else{
14     fwd(3);
15   }
16 }
```

```
1 def s1(pkt, port){
2   fwd(2);
3 }
4 def s2(pkt, port)
5   state fail(
6     flip(1/1000))
7 {
8   if fail {
9     drop;
10   } else {
11     fwd(2);
12   }
13 }
14 def s3(pkt, port){
15   fwd(3);
16 }
```

```
1 query probability(arrived@H1); // ?
```

Bayonet Program: Reliability

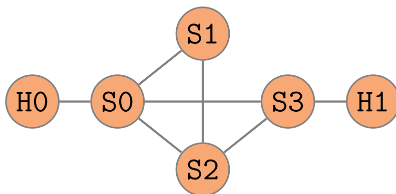
```
def h0(pkt, port){
  fwd(1);
}
def h1(pkt, port)
  state arrived(0)
{
  arrived=1;
  drop;
}
def s0(pkt, port){
  if flip(1/2){
    fwd(2);
  }else{
    fwd(3);
  }
}
```

```
def s1(pkt, port){
  fwd(2);
}
def s2(pkt, port)
  state fail(
    flip(1/1000))
{
  if fail {
    drop;
  } else {
    fwd(2);
  }
}
def s3(pkt, port){
  fwd(3);
}
```

```
query probability(arrived@H1); // ?
```

```
query probability(arrived@H1); // 0.9995
```

Bayonet Program: Congestion




```
1  topology{
2    nodes{ H0, H1, S0, S1, S2, S3 }
3    links{
4      (H0,pt1) <-> (S0,pt1),
5      (S0,pt2) <-> (S1,pt1),
6      (S0,pt3) <-> (S3,pt1),
7      (S0,pt4) <-> (S2,pt1),
8      (S1,pt2) <-> (S2,pt2),
9      (S2,pt3) <-> (S3,pt2),
10     (S3,pt3) <-> (H1,pt1)
11   }
12 }
13 queue_capacity 2;
```

Bayonet Program: Congestion

```
1  def h0(pkt, port) state pkt_count(0){
2    if pkt_count < 3 {
3      new;
4      pkt_count = pkt_count + 1;
5      fwd(1);
6    } else { drop; }
7  }
8  def s0(pkt, port){
9    if flip(1/3){ fwd(1); }
10   else if flip(1/2){ fwd(2); }
11   else{ fwd(3); }
12 }
13 def s1(pkt, port){ fwd(2); }
14 def s2(pkt, port){ fwd(3); }
15 def s3(pkt, port){ fwd(3); }
16 def h1(pkt, port) state pkt_count(0){
17   pkt_count = pkt_count + 1; drop;
18 }
19 query probability(pkt_count@H1 < 3); // ?
```

Uniform Scheduling

```
1 RunSw:=0, FwdQ:=1;
2
3 def scheduler(){
4   actions := []: (ℝ × ℝ)[]; // list of actions
5   for i in [0..k) { // k: number of nodes
6     if (Q_in@i).size() > 0 {
7       actions ~= [(RunSw, i)]; } // run node
8     if (Q_out@i).size() > 0 {
9       actions ~= [(FwdQ, i)]; // forward packet
10    }
11  }
12  n := actions.length; // pick action u.a.r.:
13  return actions[uniformInt(0,n-1)];
14 }
```

```
1 query probability(pkt_count@h1 < 3); // ?
```

Bayonet to PSI

```
1  dat s0{
2    Q_in: Queue[ℝ × ℝ], Q_out: Queue[ℝ × ℝ];
3    def s0(){
4      Q_in = Queue[ℝ × ℝ]();
5      Q_out = Queue[ℝ × ℝ]();
6    }
7    def run(){
8      (pkt, pt) = Q_in.takeFront();
9      d_pt := if flip(1/3) then 1
10             else if flip(1/2) then 2
11             else 3;
12     Q_out.pushBack(pkt, d_pt);
13   }
14 }
```

Bayonet to PSI

```
1  dat h1{
2    Q_in: Queue[ℝ × ℝ], Q_out: Queue[ℝ × ℝ];
3    pkt_count: ℝ;
4    def h1(){
5      Q_in = Queue[ℝ × ℝ]();
6      Q_out = Queue[ℝ × ℝ]();
7      pkt_count = 0;
8    }
9    def run(){
10     pkt_count = pkt_count + 1;
11     Q_in.popFront();
12   }
13 }
```

```
1  dat Network {
2    programs := [ 0 ↦ h0(), 1 ↦ s0(), ...];
3    links := {(0, 1)↦(2, 1), (2, 2)↦(3, 1), ...};
4    def scheduler(){ ... }
5    def step(){
6      (action, id) := scheduler();
7      if action == Run {
8        programs[id].run();
9      } else { // action == Fwd
10       (pkt, s_pt):=programs[id].Q_out.takeFront
            ();
11       (d_id, d_pt) := links[(id, s_pt)];
12       programs[d_id].Q_in.pushBack(pkt, d_pt);
13   } }
14   def main() {
15     repeat num_steps {
16       if !terminated() {
17         step();
18     } }
19     assert(terminated());
20     return programs[1].pkt_count<3; // query
21 } }
```

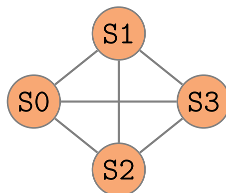
Congestion: PSI Result

The translation from Bayonet to PSI is automated:

```
1 $ bayonet congestion.bayonet > congestion.psi
2 $ psi --noboundscheck --dp --expectation \
    congestion.psi
3 𝔼[q₁] = 2663191003150725233/5997013881313296384
```

```
1 query probability(pkt_count@H1 < 3);
2   // 2663191003150725233/5997013881313296384 ≈ 0.444
```

Bayonet Program: Gossip




```
1  topology{
2    nodes { S0, S1, S2, S3 }
3    links{
4      (S0,pt1) <-> (S1,pt3),
5      (S0,pt2) <-> (S2,pt2),
6      (S0,pt3) <-> (S3,pt1),
7      (S1,pt1) <-> (S2,pt3),
8      (S1,pt2) <-> (S3,pt2),
9      (S2,pt1) <-> (S3,pt3)
10   }
11 }
```

Bayonet Program: Gossip

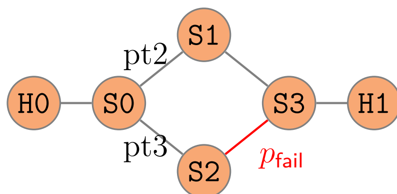
```
1  def first(pkt,port) state infected(0){
2    if infected == 0 {
3      infected = 1;
4      new;
5      fwd(uniformInt(1,3));
6    }else{ drop; }
7  }
8
9  def node(pkt,port) state infected(0){
10   if infected == 0{
11     infected = 1;
12     dup;
13     fwd(uniformInt(1,3));
14     fwd(uniformInt(1,3));
15   }else{ drop; }
16 }
17 query expectation(infected@S0 + infected@S1 +
     infected@S2 + infected@S3); // ?
```

Gossip: PSI Result

```
1 $ bayonet gossip.bayonet > gossip.psi
2 $ psi --noboundscheck --dp --expectation \
    gossip.psi
3 𝔼[q₁] = 94/27
```

```
1 query expectation(infected@S0 + ...);
2   // 94/27 ≈ 3.481
```

Bayonet Program: Reliability with Observations




```
1  topology{
2    nodes{ H0, H1, S0, S1, S2, S3 }
3    links{
4      (H0,pt1) <-> (S0,pt1),
5      (S0,pt2) <-> (S1,pt1),
6      (S0,pt3) <-> (S2,pt1),
7      (S1,pt2) <-> (S3,pt1),
8      (S2,pt2) <-> (S3,pt2),
9      (S3,pt3) <-> (H1,pt1)
10   }
11 }
```

Uniform Scheduling. Host H0 sends 3 packets.

```
1  packet_fields{ id }
2  def h0(pkt,port) state pkt_count(0){
3    if pkt_count < 3 {
4      new;
5      pkt_count = pkt_count + 1;
6      pkt.id = pkt_count;
7      fwd(1);
8    } else { drop; }
9  }
10 def s0(pkt,port) state strategy(4){
11   if strategy == 4{
12     strategy = uniformInt(1,2);
13     if strategy == 2{
14       strategy = strategy + flip(1/2);
15   } }
16   if strategy == 1{
17     if flip(1/2){  fwd(2); }else{ fwd(3); }
18   }else{
19     fwd(strategy);
20   }
21 }
```

Observing the Packet Sequence

- We observe a particular sequence.
- Compute posterior distribution on behaviour of node s0

```
def h1(pkt,port) state num_arrived(0){
  num_arrived = num_arrived + 1;
  observe(num_arrived-1 < sequence.length);
  observe(pkt.id == sequence[num_arrived-1]);
  drop;
}
post_observe num_arrived@H1 == sequence.length;
```

```
sequence := [1, 2, 3]; // in-order
query strategy@s0 == 1; // ? (random)
query strategy@s0 == 2; // ? (det. S1)
query strategy@s0 == 3; // ? (det. S2)
```

In-order Packet Arrival: Distribution

```
1 $ bayonet gossip.bayonet > gossip.psi
2 $ psi --noboundscheck --dp --expectation \
    reliability-observe-1-2-3.psi
3 𝔼[q₁, q₂, q₃] = (41922792469/95643630613, \
4                  26873856000/95643630613, \
5                  26846982144/95643630613)
```

```
1 sequence := [1, 2, 3];
2 query strategy@s0 == 1; // ≈ 0.438322 (random)
3 query strategy@s0 == 2; // ≈ 0.280979 (det. S1)
4 query strategy@s0 == 3; // ≈ 0.280698 (det. S2)
```

Reordered Packets

```
1 sequence := [2, 1, 3];
2 query strategy@s0 == 1; // ?
3 query strategy@s0 == 2; // ?
4 query strategy@s0 == 3; // ?
```

Reordered Packets: Distribution

```
1 sequence := [2, 1, 3];
2 query strategy@s0 == 1; // =1 (random)
3 query strategy@s0 == 2; // =0 (det. S1)
4 query strategy@s0 == 3; // =0 (det. S2)
```

Nothing Arrives

```
1 sequence := [];
2 query strategy@s0 == 1; // ?
3 query strategy@s0 == 2; // ?
4 query strategy@s0 == 3; // ?
```

Nothing Arrives: Distribution

```
1 sequence := [];
2 query strategy@s0 == 1; // = 0.2 (random)
3 query strategy@s0 == 2; // = 0.0 (det. S1)
4 query strategy@s0 == 3; // = 0.8 (det. S2)
```

Bayonet: Summary

- Networks exhibit probabilistic behaviors
- Can model in a *probabilistic programming language*
- Use existing solvers for inference. Benefits:
  - Do not reinvent the wheel.
  - Can use to test a specialized solution.
  - Provide benchmarks to general tools.