An architecture for metarouting

John N. Billings, Philip J. Taylor, Timothy G. Griffin

University of Cambridge

RiNG Workshop, Madrid, December 2007

Metarouting

- Define a metalanguage in which we can write new routing protocols

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions
- Compile to efficient code

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions
- Compile to efficient code
- BGP in a few pages?

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions
- Compile to efficient code
- BGP in a few pages?
- Create a tool for network researchers / operators

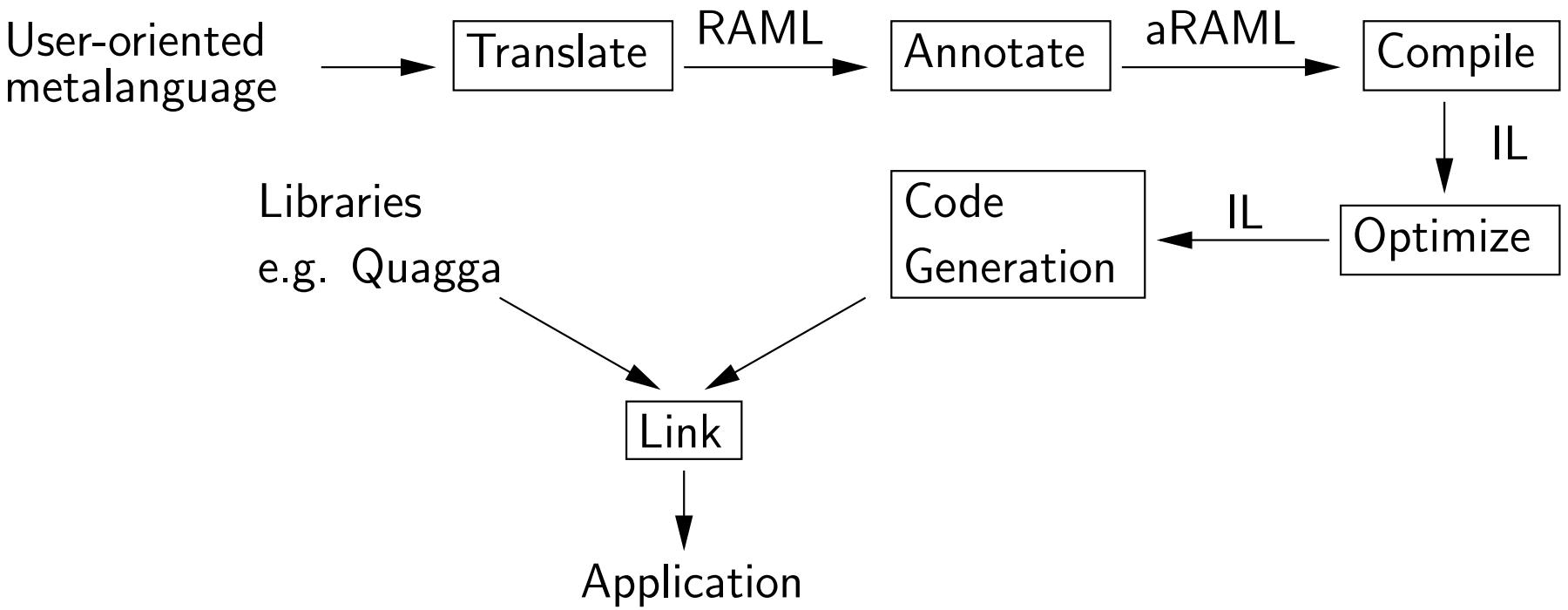
Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions
- Compile to efficient code
- BGP in a few pages?
- Create a tool for network researchers / operators
- Long-term project, just getting started

Metarouting

- Define a metalanguage in which we can write new routing protocols
- Mechanise the 'boiler-plate' for routing protocols e.g. wire formats, data-structures, policy application, metric comparisons
- *Automatically* check correctness conditions
- Compile to efficient code
- BGP in a few pages?
- Create a tool for network researchers / operators
- Long-term project, just getting started
- First ever demonstration of working code

System overview



Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>
- Policy application: add distance, minimise bandwidth

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>
- Policy application: add distance, minimise bandwidth
  - <dist=1, bw=60> . <dist=10, bw=50>

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>
- Policy application: add distance, minimise bandwidth
  - <dist=1, bw=60> . <dist=10, bw=50>
  - $\Rightarrow$ <dist=11, bw=50>

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>
- Policy application: add distance, minimise bandwidth
  - <dist=1, bw=60> . <dist=10, bw=50>
  - $\Rightarrow$ <dist=11, bw=50>
- Automatically infer *monotonicity*

Example RAML specification

```
preorder-semigroup dist_bw =
  lex_product <dist : positive_integer_lte_plus,
               bw   : positive_integer_gte_min>
```

- Defines *order* and *policy application* function over metrics
- Metrics: <dist=*d*, bw=*b*>
- Order: compare distances (smaller is better), tie-break on bandwidth (larger is better)
  - <dist=10, bw=50> prefered to <dist=20, bw=100>
  - <dist=10, bw=50> prefered to <dist=10, bw=40>
- Policy application: add distance, minimise bandwidth
  - <dist=1, bw=60> . <dist=10, bw=50>
  - $\Rightarrow$ <dist=11, bw=50>
- Automatically infer *monotonicity*
- Use with generalised Dijkstra (e.g. OSPF, IS-IS) or vectoring mechanism (e.g. RIP, BGP)

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance
- Policy application as before

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance
- Policy application as before
- *Increasing* (*not* monotonic).

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance
- Policy application as before
- *Increasing* (*not* monotonic).
- Can only use with vectoring (e.g. RIP, BGP). [Sobrinho03]

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance
- Policy application as before
- *Increasing* (*not* monotonic).
- Can only use with vectoring (e.g. RIP, BGP). [Sobrinho03]
- Can count to infinity

Example RAML specification (2)

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Order: compare bandwidth, tie-break on distance
- Policy application as before
- *Increasing* (*not* monotonic).
- Can only use with vectoring (e.g. RIP, BGP). [Sobrinho03]
- Can count to infinity
- Will demonstrate later!

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

- Order: as before, but additionally tie-break on path length

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

- Order: as before, but additionally tie-break on path length
- Policy application: as before, but add on new path element

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

- Order: as before, but additionally tie-break on path length
- Policy application: as before, but add on new path element
- Additional constraint: no duplicate path elements (cf. BGP)

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

- Order: as before, but additionally tie-break on path length
- Policy application: as before, but add on new path element
- Additional constraint: no duplicate path elements (cf. BGP)
- No counting to infinity

Example RAML specification (3)

```
preorder-semigroup bw_dist_path =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus,
               path : router_path>
```

- Order: as before, but additionally tie-break on path length
- Policy application: as before, but add on new path element
- Additional constraint: no duplicate path elements (cf. BGP)
- No counting to infinity
- Lots of other possible specifications...

Example of generated code

```
static int mrc_int_cmp(mrc_int_t x, mrc_int_t y) {
  return y - x;
}

static int mrc_string_cmp(mrc_string_t x, mrc_string_t y) {
  int res;
  if (x == y)
    res = 0;
  else /* Lexicographic ordering */
    res = strcmp(x->value, y->value);
  return res;
}

static int mrc_list_cmp(mrc_slist_t x, mrc_slist_t y) {
  int res;
  if (x == y) /* Physical equality */
    res = 0;
  else {
    for(; x && y; x = x->next, y = y->next)
      if ((res = mrc_string_cmp(x->value, y->value))) /* Elements non-equal */
        goto end;
    if (x == NULL) {
      if (y == NULL) /* Structural equality */
        res = 0;
      else /* x is prefix of y */
        res = -1;
    }
    else /* y is prefix of x */
      res = 1;
  }
end: return res;
}
```

Generalised routing algorithms

Open research topic!

- Take existing routing protocol e.g. RIP, BGP, OSPF, IS-IS

Generalised routing algorithms

Open research topic!

- Take existing routing protocol e.g. RIP, BGP, OSPF, IS-IS
- Abstract metric-specific operations behind API

Generalised routing algorithms

Open research topic!

- Take existing routing protocol e.g. RIP, BGP, OSPF, IS-IS
- Abstract metric-specific operations behind API
- e.g. comparisons, policy application, printing, marshaling, ...

Generalised routing algorithms

Open research topic!

- Take existing routing protocol e.g. RIP, BGP, OSPF, IS-IS
- Abstract metric-specific operations behind API
- e.g. comparisons, policy application, printing, marshaling, ...
- Currently: generalised Quagga RIP implementation

Generalised routing algorithms

Open research topic!

- Take existing routing protocol e.g. RIP, BGP, OSPF, IS-IS
- Abstract metric-specific operations behind API
- e.g. comparisons, policy application, printing, marshaling, ...
- Currently: generalised Quagga RIP implementation
- Result: generalised soft-state, distance vector protocol.

API for generalised RIP

```
metric_t metric_parse(const char*);
size_t metric_print(char*, size_t, metric_t);

metric_t metric_copy(metric_t);
void metric_free(metric_t);

size_t metric_marshal(void*, size_t, metric_t);
metric_t metric_unmarshal(const void*, size_t);

metric_t metric_infinity(void);
metric_t policy_apply(policy_t, metric_t);

int metric_is_better(metric_t, metric_t);
int metric_is_infinity(metric_t);
int metric_is_equal(metric_t, metric_t);
```

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing
  - Decreased memory usage

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing
  - Decreased memory usage
  - Fast comparsions using pointer equality checks

Efficient code for metrics

Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing
  - Decreased memory usage
  - Fast comparsions using pointer equality checks
  - Side-effect: immutable metrics

Efficient code for metrics

Open research topic!

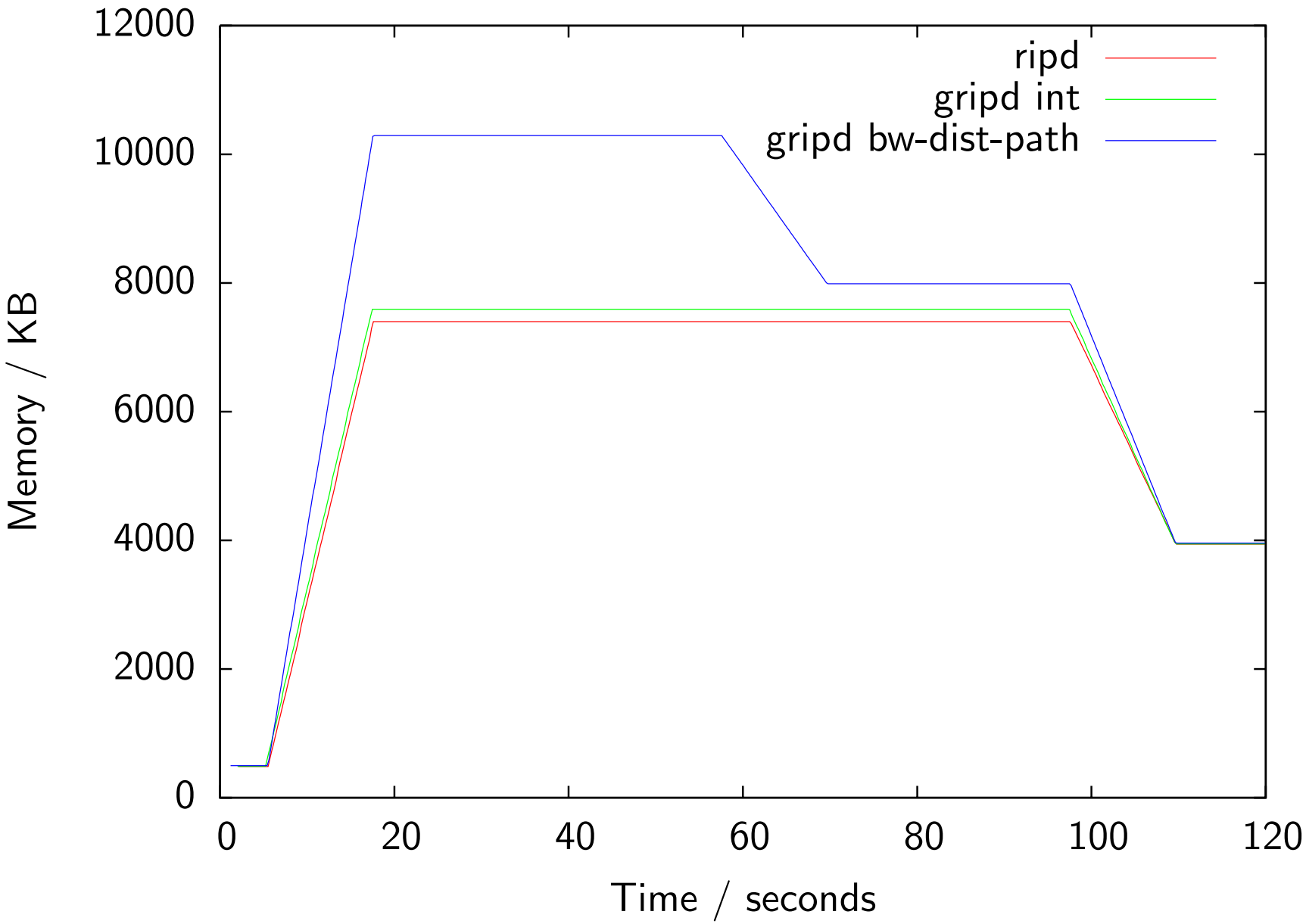
- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing
  - Decreased memory usage
  - Fast comparsions using pointer equality checks
  - Side-effect: immutable metrics
- Clean code that can be optimised by the C compiler

Efficient code for metrics

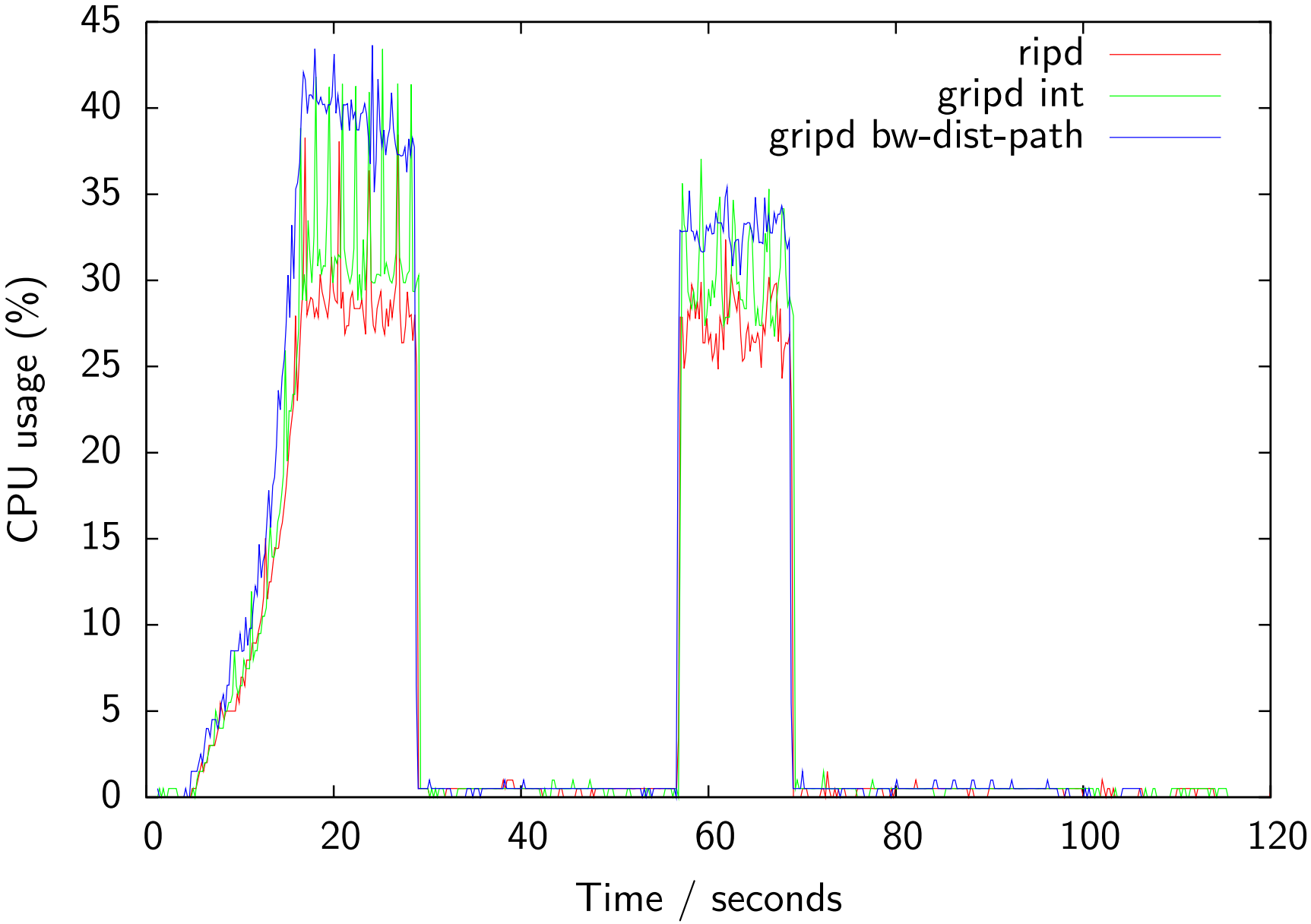
Open research topic!

- 3 areas: time, memory, bandwidth
- 'Reasonable' data-structures e.g. red-black trees for sets
- Only store metrics on heap if they don't fit within a word
- Maximise sharing
  - Decreased memory usage
  - Fast comparsions using pointer equality checks
  - Side-effect: immutable metrics
- Clean code that can be optimised by the C compiler
- Goal: tunable tradeoff between time and memory

Performance: memory



Performance: CPU



Simulation

- Use QEMU to emulate multiple routers on single machine

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces
  - Cheap!

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces
  - Cheap!
- Cons:

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces
  - Cheap!
- Cons:
  - Timing is unrealistic (shared CPU, network latency)

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces
  - Cheap!
- Cons:
  - Timing is unrealistic (shared CPU, network latency)
  - No heterogeneity of hardware or code

Simulation

- Use QEMU to emulate multiple routers on single machine
- Run real OS and routing code
- Communicate over virtual networks
- Pros:
  - Test generated protocols (almost) 'for real'
  - Behaviour should be identical to corresponding physical network (modulo timing)
  - Easy to configure
  - Event traces
  - Cheap!
- Cons:
  - Timing is unrealistic (shared CPU, network latency)
  - No heterogeneity of hardware or code
- What do you do?

Future work

- Expressiveness e.g. how do we model EIGRP in general?

Future work

- Expressiveness e.g. how do we model EIGRP in general?
- How does forwarding fit into the model of routing?

Future work

- Expressiveness e.g. how do we model EIGRP in general?
- How does forwarding fit into the model of routing?
- Generalise OSPF and BGP implementations

Future work

- Expressiveness e.g. how do we model EIGRP in general?
- How does forwarding fit into the model of routing?
- Generalise OSPF and BGP implementations
- Redistribution

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>
- Apply policy <10,1>: <10,51> $\not\leq$ <10,2>

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>
- Apply policy <10,1>: <10,51> $\not\leq$ <10,2>

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>
- Apply policy <10,1>: <10,51> $\not\leq$ <10,2>
- Involves 'counting to infinity'

Demonstration: algebra

```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

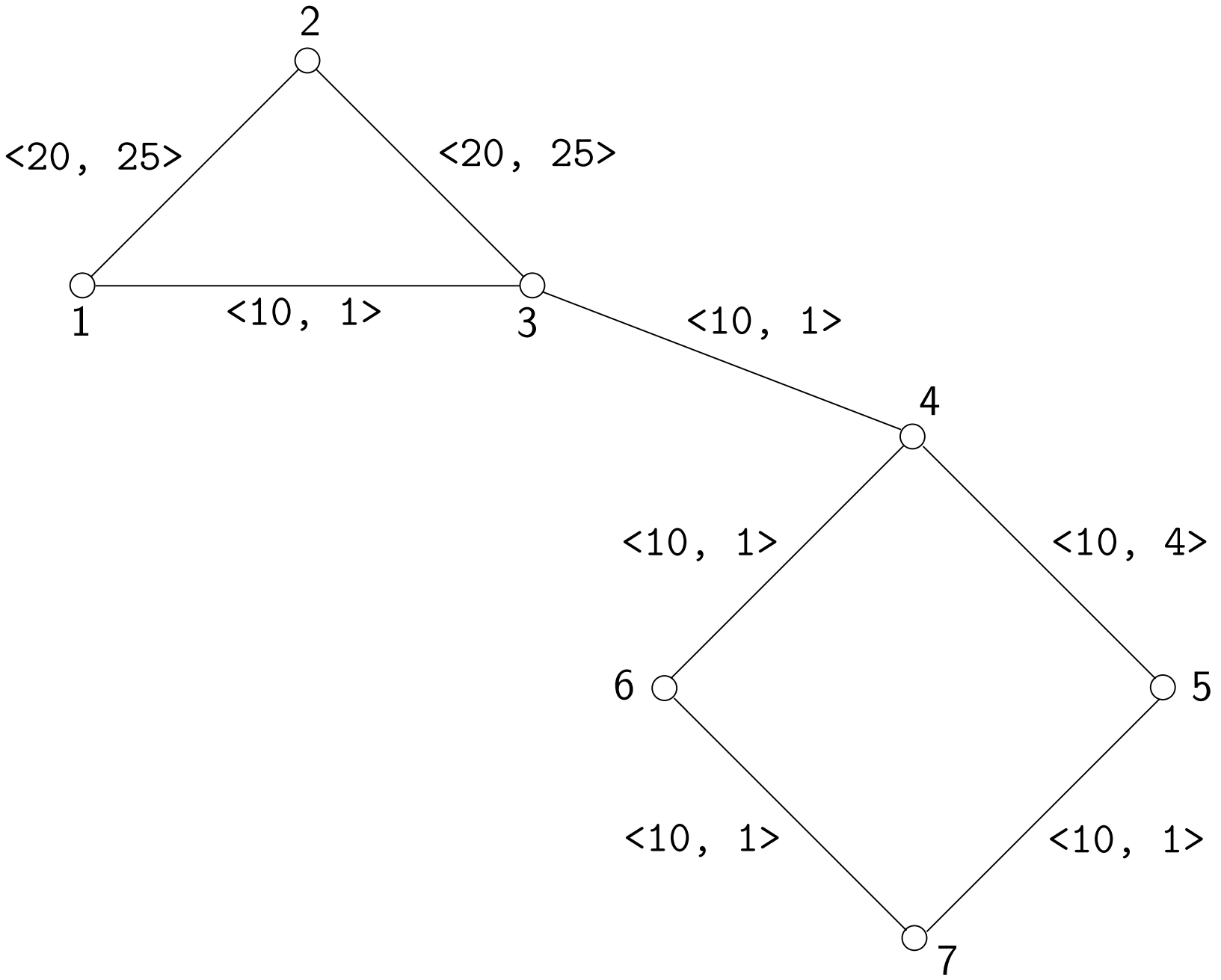
- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>
- Apply policy <10,1>: <10,51> $\not\leq$ <10,2>
- Involves 'counting to infinity'
- Trace from real code running on virtual network

Demonstration: algebra

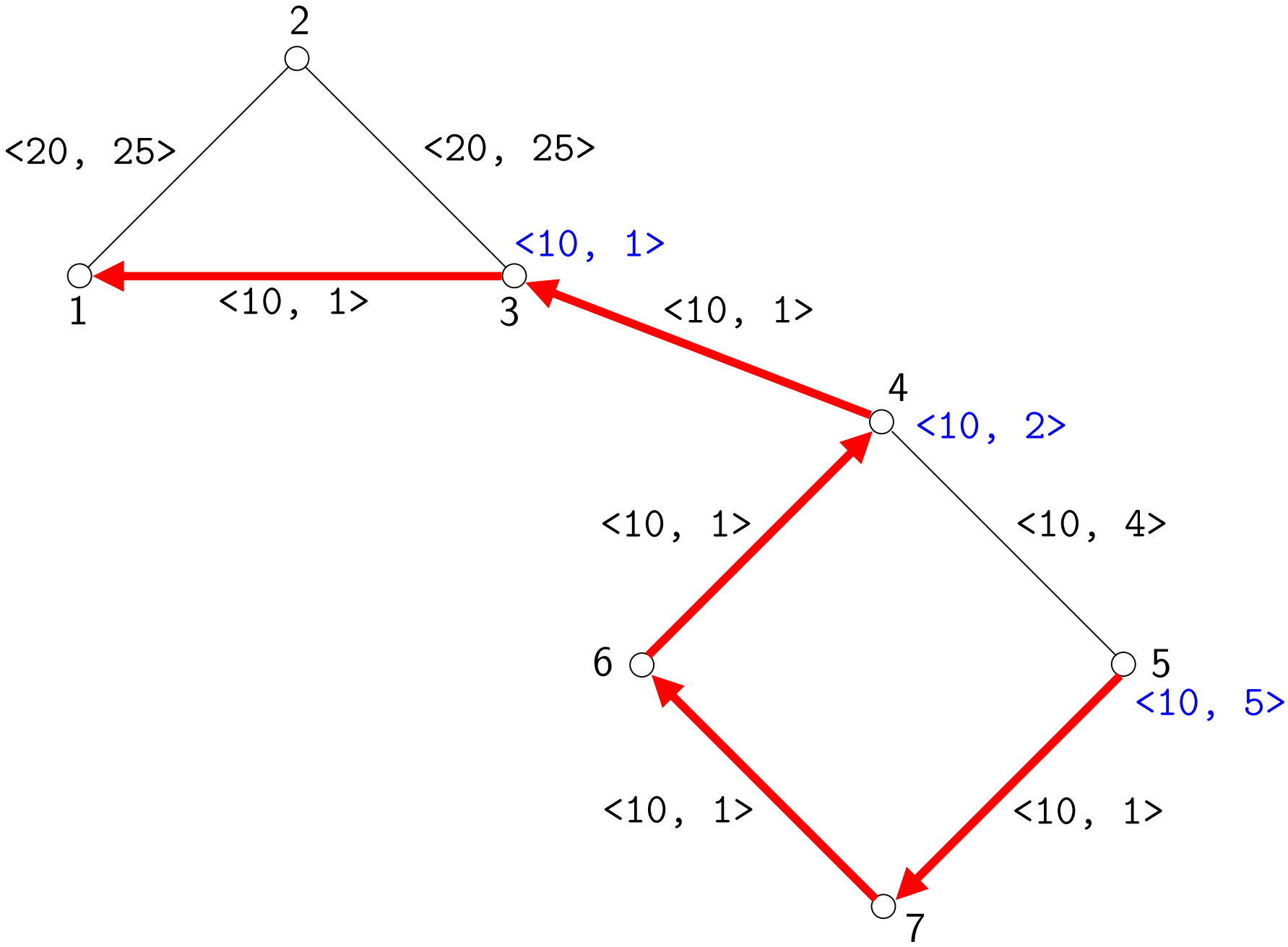
```
preorder-semigroup bw_dist =
  lex_product <bw   : positive_integer_gte_min,
               dist : positive_integer_lte_plus>
```

- Algebra is non-monotonic
- Monotonicity: $m_1 \leq m_2 \Rightarrow p(m_1) \leq p(m_2)$
- <40,50> $\leq$ <10,1>
- Apply policy <10,1>: <10,51> $\not\leq$ <10,2>
- Involves 'counting to infinity'
- Trace from real code running on virtual network
- (Adjust timings, disable triggered updates)

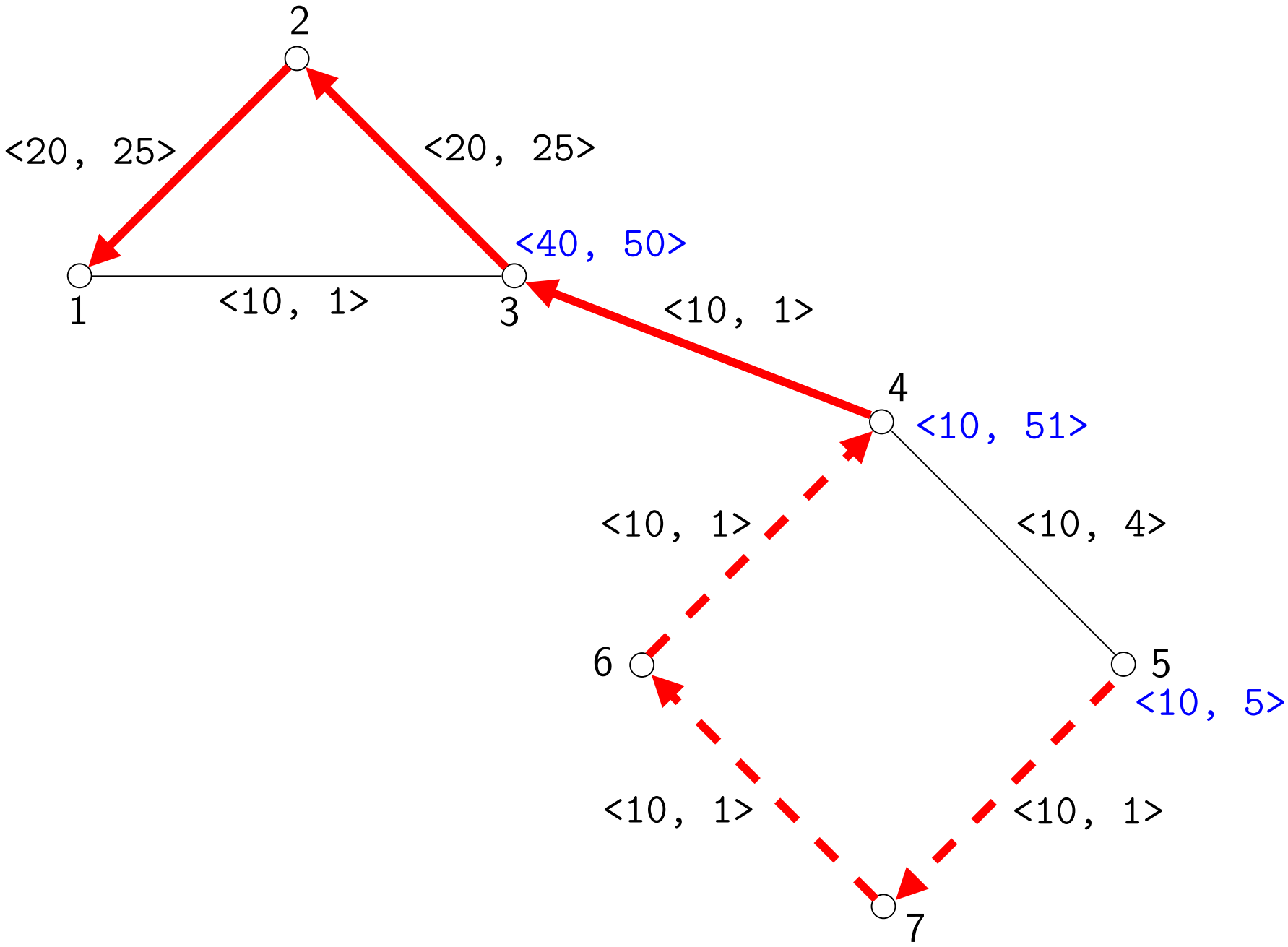
Demonstration: topology (1)



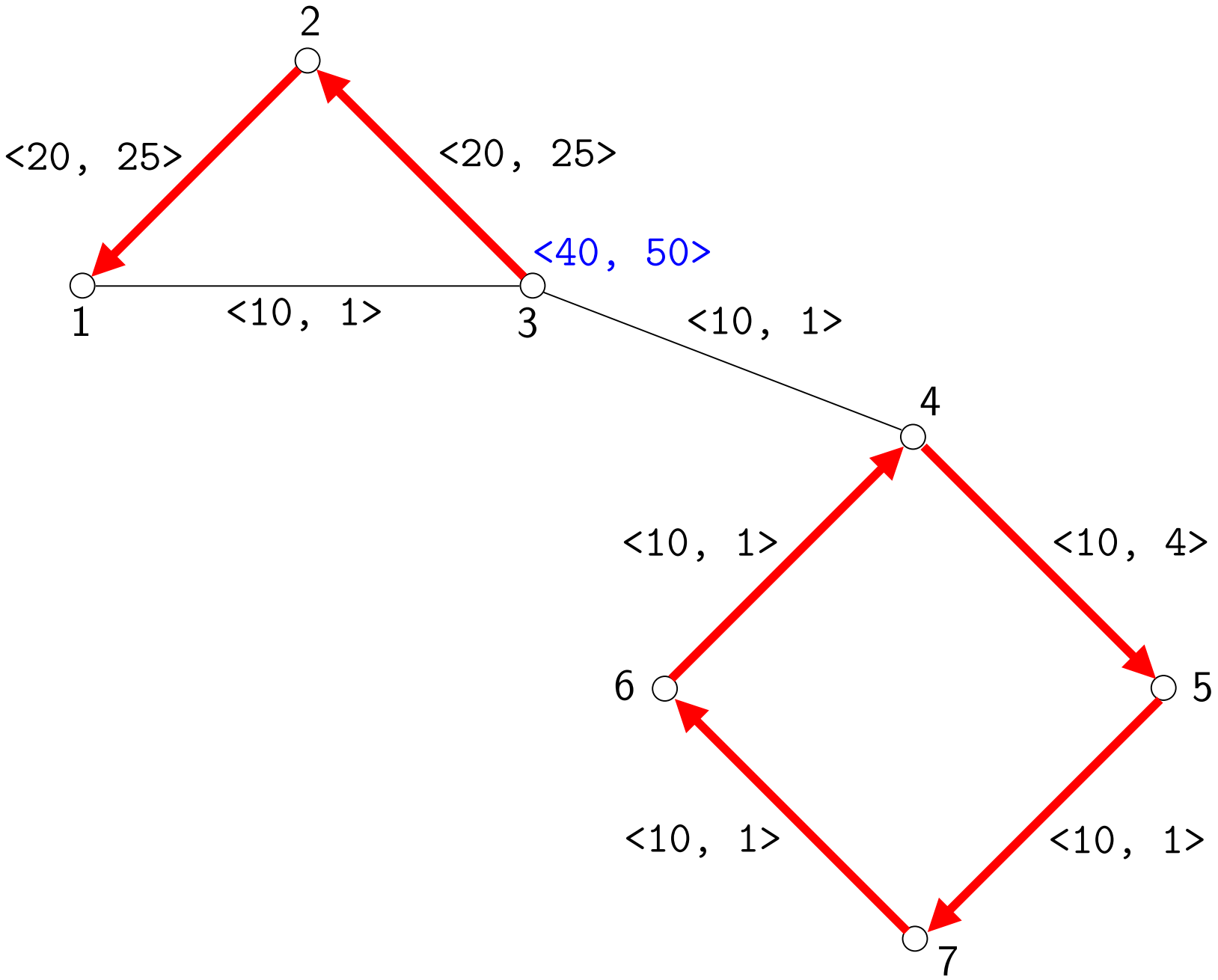
Demonstration: topology (2)



Demonstration: topology (3)



Demonstration: topology (4)



Demonstration: topology (5)