Privacy Preserving Distance-Vector Routing

or How to Distribute Routing Computations without Distributing Routing Information

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
School of Mathematical Sciences
University of Adelaide

Joint work with Yin Zhang, Olaf Maennel, and Miro Kraetzl

Link-State Routing

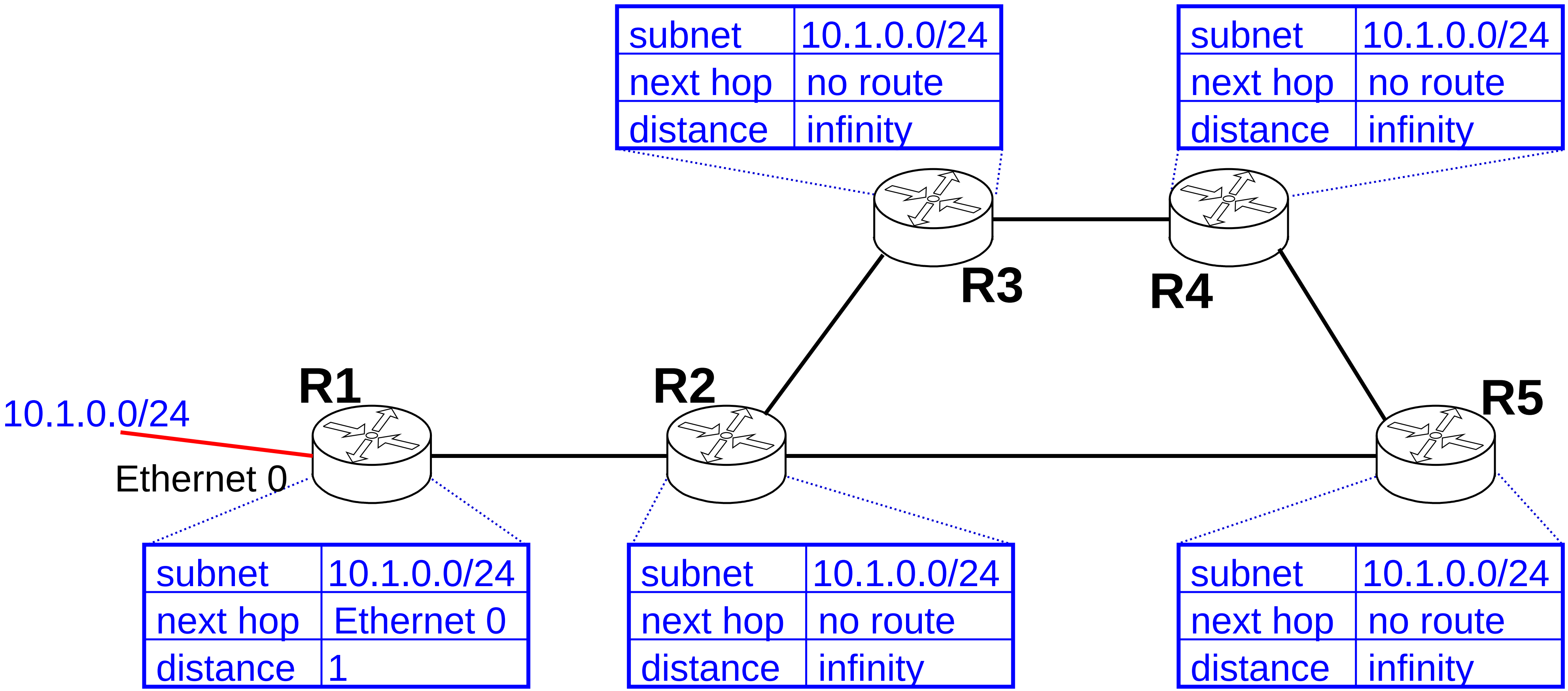
- Link-state: flood
  - topology
  - routing information (e.g. metrics)

  all nodes learn everything, and can run Dikstra independently
- Why not use this for BGP?
  - scalability
  - everyone learns everything
    - all the gory details of routing policies
- So we use path-vector
  - distance vector is a little easier for me

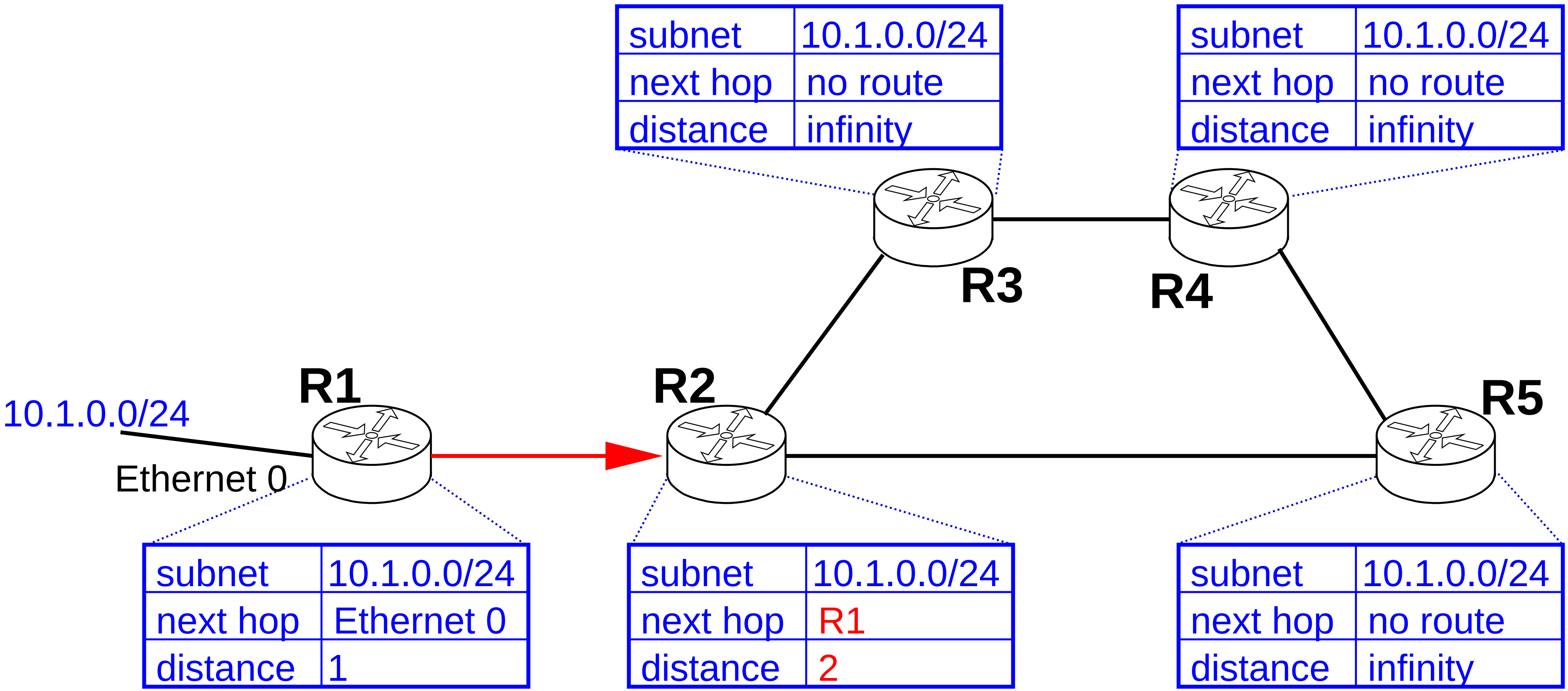
Distance-vector routing

- How it works
  - Each router has its own set of "best routes"
  - tell neighbours about your routes
  - they choose their own, and continue the process
  - "routing by rumour"
- Why is it good?
  - hope for some "compression"
    - only send best routes
  - some information hiding
    - don't learn full topology

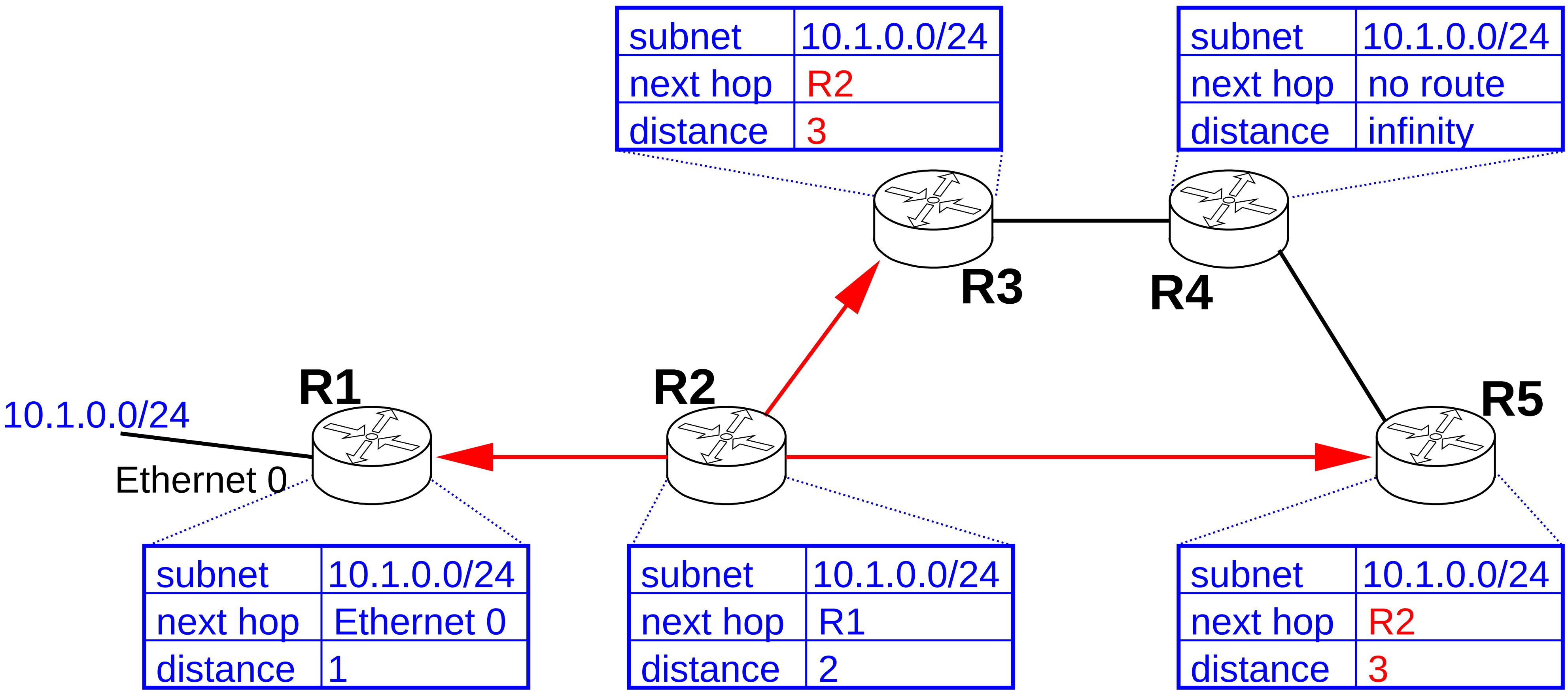
Distance Vector example



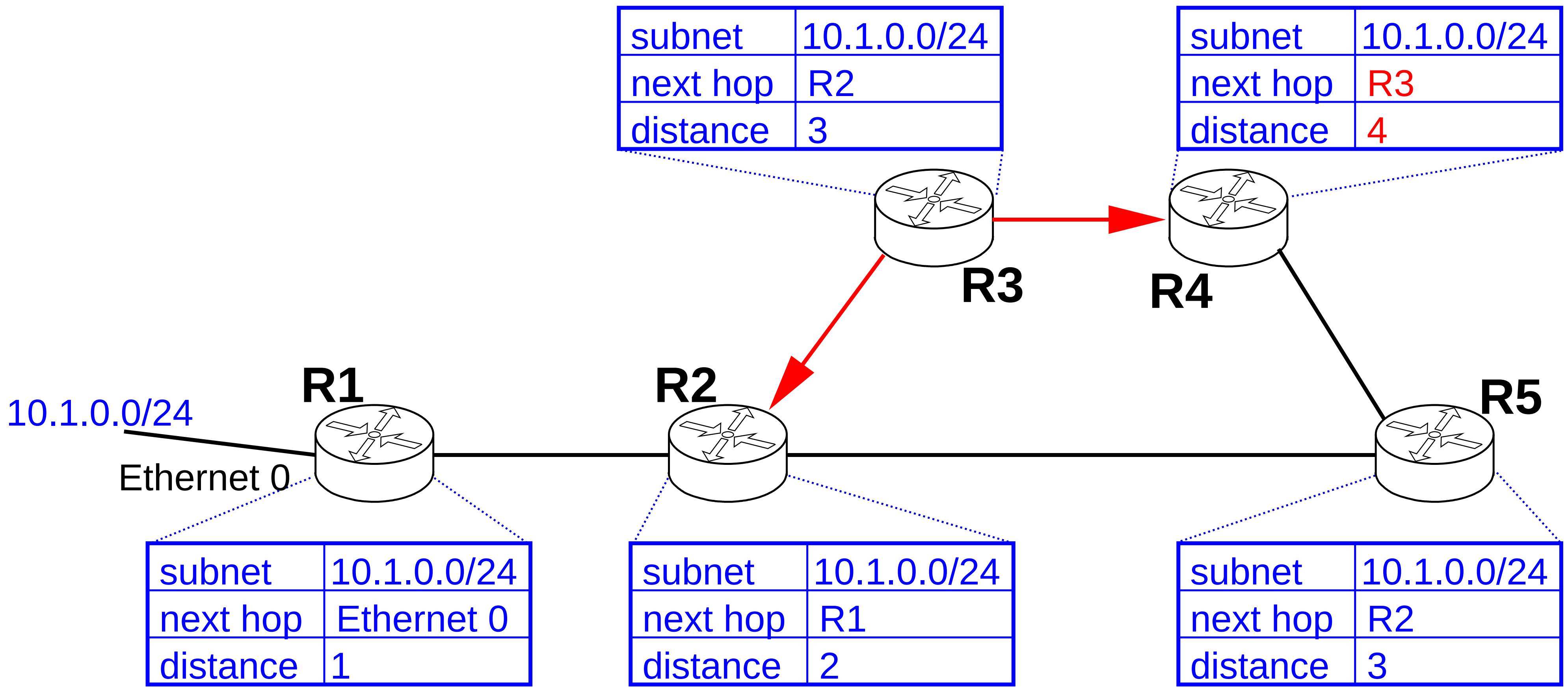
Distance Vector example



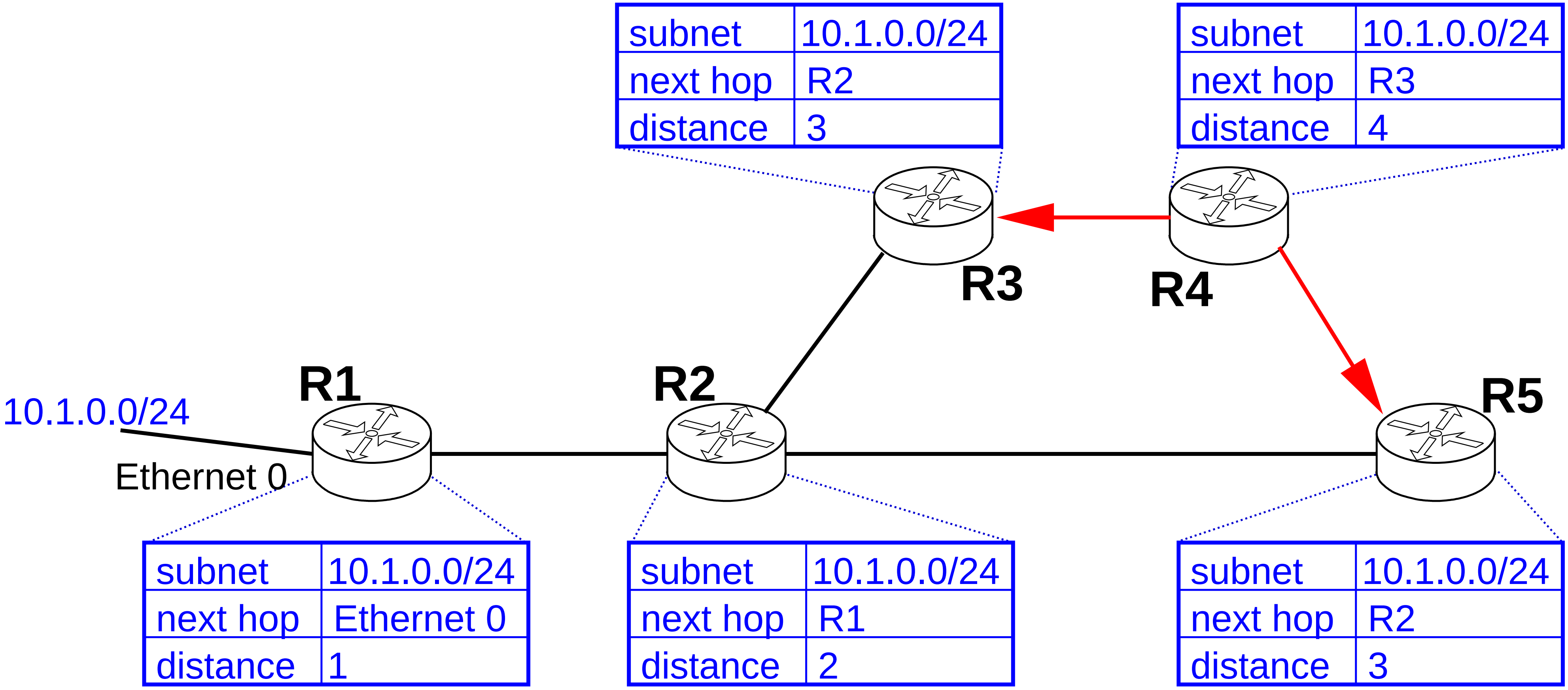
Distance Vector example



Distance Vector example



Distance Vector example



Information hiding

- some info is hidden
  - like actual topology
- some information is revealed
  - distances along the different alternative paths
- some information can be inferred
  - hop counts in RIP can tell you a lot about topology, particularly when seen from a few viewpoints

Information hiding

- what if you had a network where you didn't trust all routers
  - perhaps some might be compromised
    - e.g., military networks might worry about this
  - ad hoc networks
    - who knows who is using the network?
  - some nodes aren't fully trusted
    - e.g., Australia and other countries run joint miltary operations, but do they really trust each other?
- Scientia Potentia Est (Francis Bacon, Meditations)
  - increased network knowledge enables other attacks

Similar problems elsewhere

- The Center for Disease Control and Prevention (CDC) who have to detect new health threats
  - need data from
    - hospitals
    - insurance companies, airlines, ...
    - NGOs (e.g. charities)
    - other government bodies
  - data is
    - proprietary (e.g. insurance risks)
    - protected by privacy legislation
  - data-mining community has developed solutions
    - secure-distributed computing [3, 4, 5]
    - privacy-preserving data-mining [6, 7]

Trusted third party

- simple answer: a trusted third party
  - independent party (e.g. with no vested interest)
  - trusted by all routers
  - collects data, and determines routes and shares the results
- problems:
  - hard to find such parties
  - introduce a central point of failure
  - doesn't scale

A Couple of problems

Well known problems in secure distributed computing

- Dining cryptographers
- Millionaire problem
  - Bill Gates and Warren Buffet are trying to decide who should put more money into the Gates foundation $(*)$
    - they want to know who is richer
  - But they are feeling rather secretive, and don't want to reveal their true wealth.
  - how can they decide?

$(*)$ – no real millionaires were harmed in the production of these slides

Primitives

There are some generic techniques that can help us out

- Secure Distributed Summation (SDS)
- Secure Distributed Dot Product (SDP)
- Oblivious Transfer (OT)
- Secure Distributed Minimum (SDM)

Honest but curious model

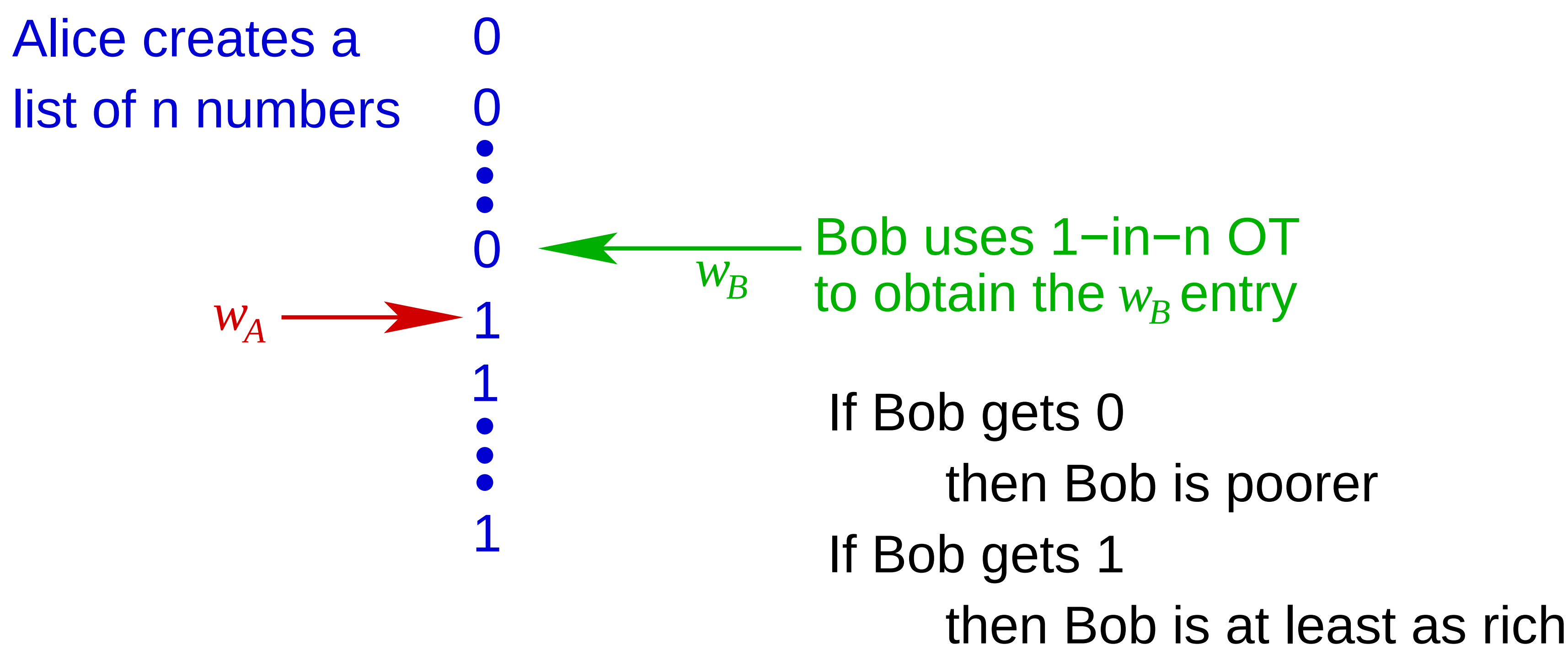
- parties could corrupt the result by changing inputs
- type of calc. has implicit assumption of honesty
  - let us extend this
- "Honest but curious" security model
  - **honest:** honestly follow protocol
  - **curious:** may perform more operations to try and learn more information (than they were supposed to learn)
- we do allow colluding coalitions
- there are stronger approaches we could incorporate
  - honest majority
  - verifiable secrets

Oblivious transfer [4, 5]

- there are various versions
- consider 1-in-n Oblivious Transfer (OT)
  - Alice has a list of numbers $\{a_1, a_2, \ldots, a_n\}$
  - Bob has an index β
  - Bob wants to learn a_β
  - Alice must not learn β, and Bob must not learn a_i for any $i \neq \beta$.
- Bob learns exactly one item from Alice's list, without Alice learning which item Bob discovered.

Applications

- the millionaires problem
  - more generically: calculating a minimum
- Assume Alice has wealth $w_A \in [1, n]$, and Bob has $w_B \in [1, n]$, where n is known to both



Secure Multi-party Minimum

The problem is reminiscent of the "Cocaine Auction"

- characteristics of our problem are a little different
- we suggest a somewhat different protocol

Requirements:

- Have one central node C that learns which of the participants has the minimal value.
- Participants (other than C) learn nothing, not even how many other participants there are.
- C learns nothing except who the participants are, and which set of these have the minimal value.
  - learns the complete set

Secure Multi-party Minimum

1. The nodes $\{p_1, \ldots, p_N\}$ choose a prime number $n \neq N$, and agree on a random vector $\mathbf{r} = (r_0, r_1, \ldots, r_{K-1})$ where r_i is uniformly distributed over $\{0, \ldots, n-1\}$. This could be accomplished by simply designating one of the p_i as the generator, or each generating one value in turn.
2. Each node p_i also generates a second random vector: $\mathbf{s}_i = (0, \ldots, 0, s^{(i)}_{x_i+1}, s^{(i)}_{x_i+2}, \ldots, s^{(i)}_{K-1})$ and creates the following vector $\mathbf{v}_i = (r_0, \ldots, r_{x_i}, s^{(i)}_{x_i+1}, s^{(i)}_{x_i+2}, \ldots, s^{(i)}_{K-1})$, i.e.

$$v_i^{(k)} = \begin{cases} r_k & \text{if } k \leq x_i, \\ s_k^{(i)} & \text{otherwise.} \end{cases}$$

3. The nodes $\{p_1, \ldots, p_N\}$ perform a SDS via C to add the $v_i^{(k)}$, and they tell C the sum.
4. C generates a random number w, and adds the sum of the $v_i^{(k)}$ and divides by N mod n, to get

$$\mathbf{V} = w + \frac{1}{N} \sum_{i=1}^{N} \mathbf{v}_i \bmod n$$

   Hence $\mathbf{V} = (w + r_1, w + r_2, \ldots, w + r_x, \cdot, \ldots, \cdot)$ where $x = \min_i \{x_i\}$.

5. Each node p_i does a 1-in-K oblivious transfer to retrieve the x_ith element of the vector $\mathbf{V}$ from C.
6. Node p_i computes $t_i = r_{x_i} - V_i \bmod n$ and sends t_i to C.
7. If $t_i = w \bmod n$, then C decides (with probability $1/n$ of being correct) that p_i has the minimum value.

Secure RIP (SRIP)

- routers advertise "reachability" of destinations to neigbours
  - indicates that it has a path to the destination
  - no information about the path is revealed
- when you are told of more than one possible path
  - run a SDM across the possible next-hops
  - given the shortest-path next-hop tells you its distance to the destination

SRIP leakage

- Information is leakage by SRIP
  - length of the shortest path
- this is less than RIP
  - in RIP, you learn the length of all paths
  - but during convergence of SRIP, you can might change paths, and get to learn more than one best path

SRIP++

- Origin node that originally advertises a destintion adds a random number to the distance to the destination
  - so no-one learns actual distances in the network
  - still leaks relative distances

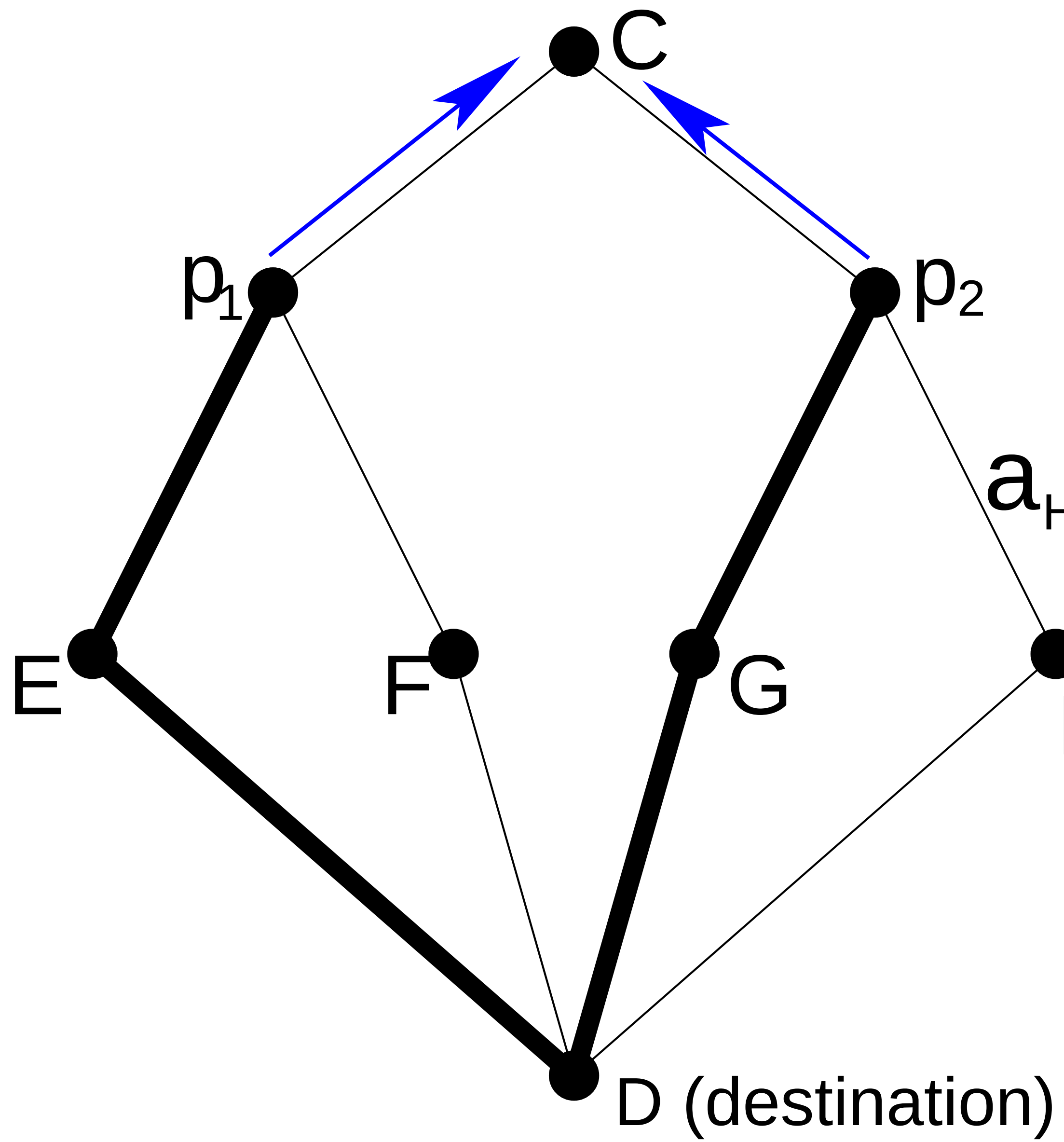
Secure Transitive RIP (STRIP)

1. a node D advertises a "destination" to its neighbours
2. when a node C hears some set of announcements of a path to a destination, it initiates a "shortest-path" computation.
   (a) it sends a request message to each neighbour that has advertised a route to that destination (label these neighbours $p_1, \ldots, p_N$).
   (b) each node that receives such a request forwards it to its next hop to the destination
   (c) the origin node D generates a random number R (generated once for each unique computation), and adds m_i the metrics to R for each message
   (d) as the reponse is passed back to p_i, the intermediate nodes add their metrics.
   (e) the neighbours of C tell A that they are ready to perform a computation. The peers p_i of C each have a value

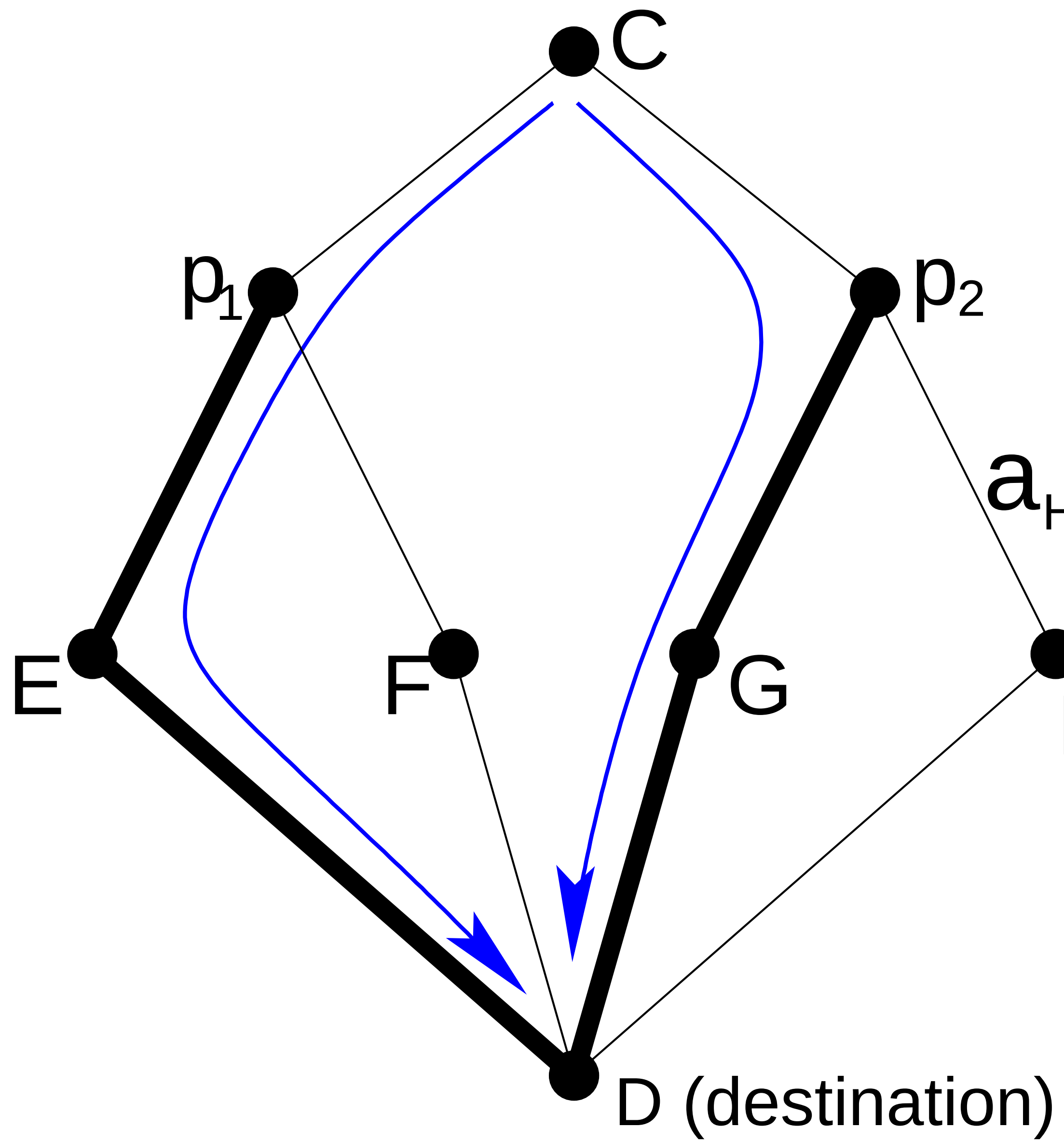
$$x_i = R + \sum_{j: j \in \mathcal{P}_i} m_j + m_i$$

   where $\mathcal{P}_i$ is the set of links along the path from node D to p_i, and m_i is the metric value on the link between C and p_i.
   (f) when C initiates a SDM operation across the N peers. The minimum of x_i will also be the minimum of $\sum_{j: j \in \mathcal{P}_i} m_j + m_i$.

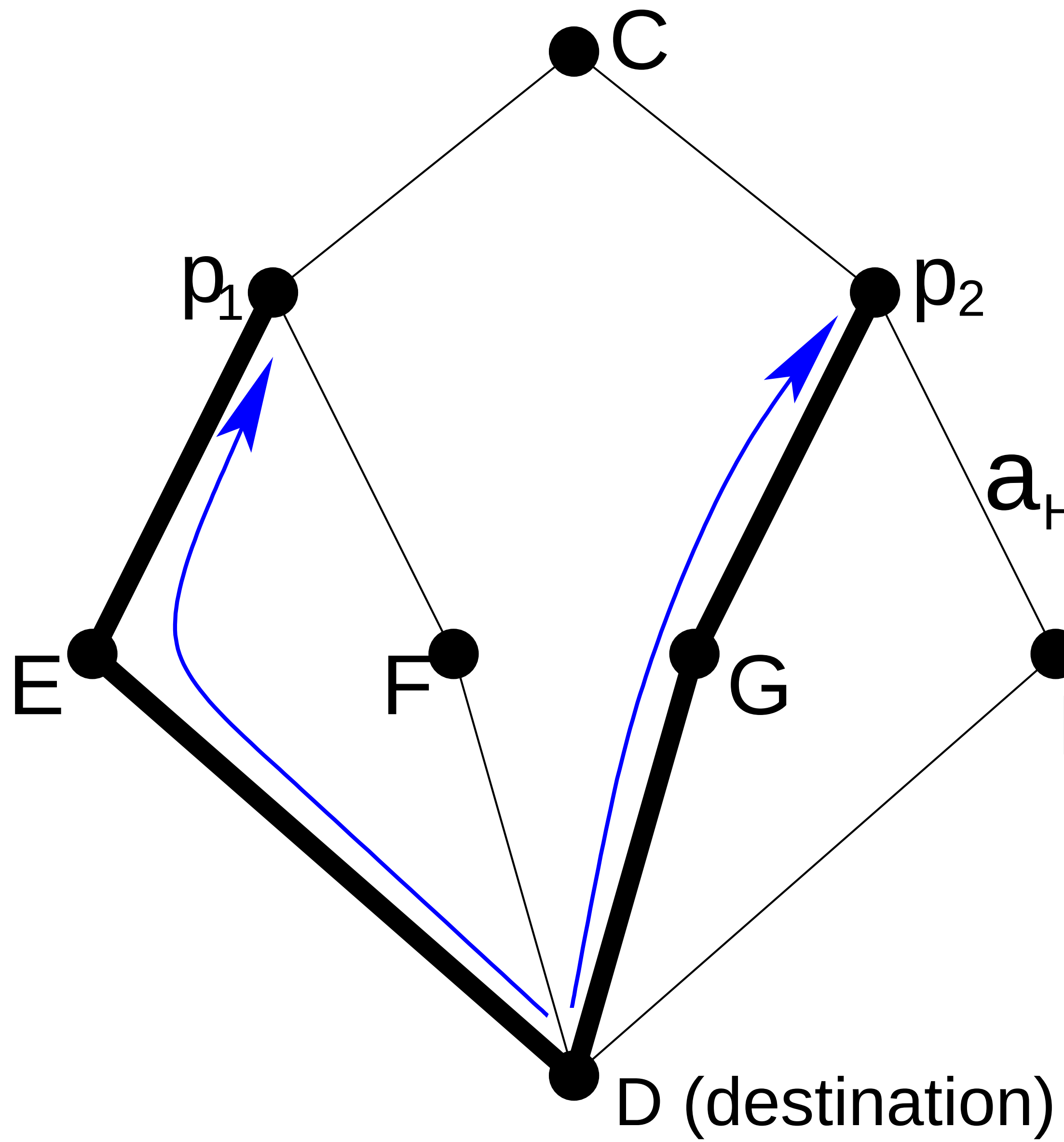
STRIP step 1



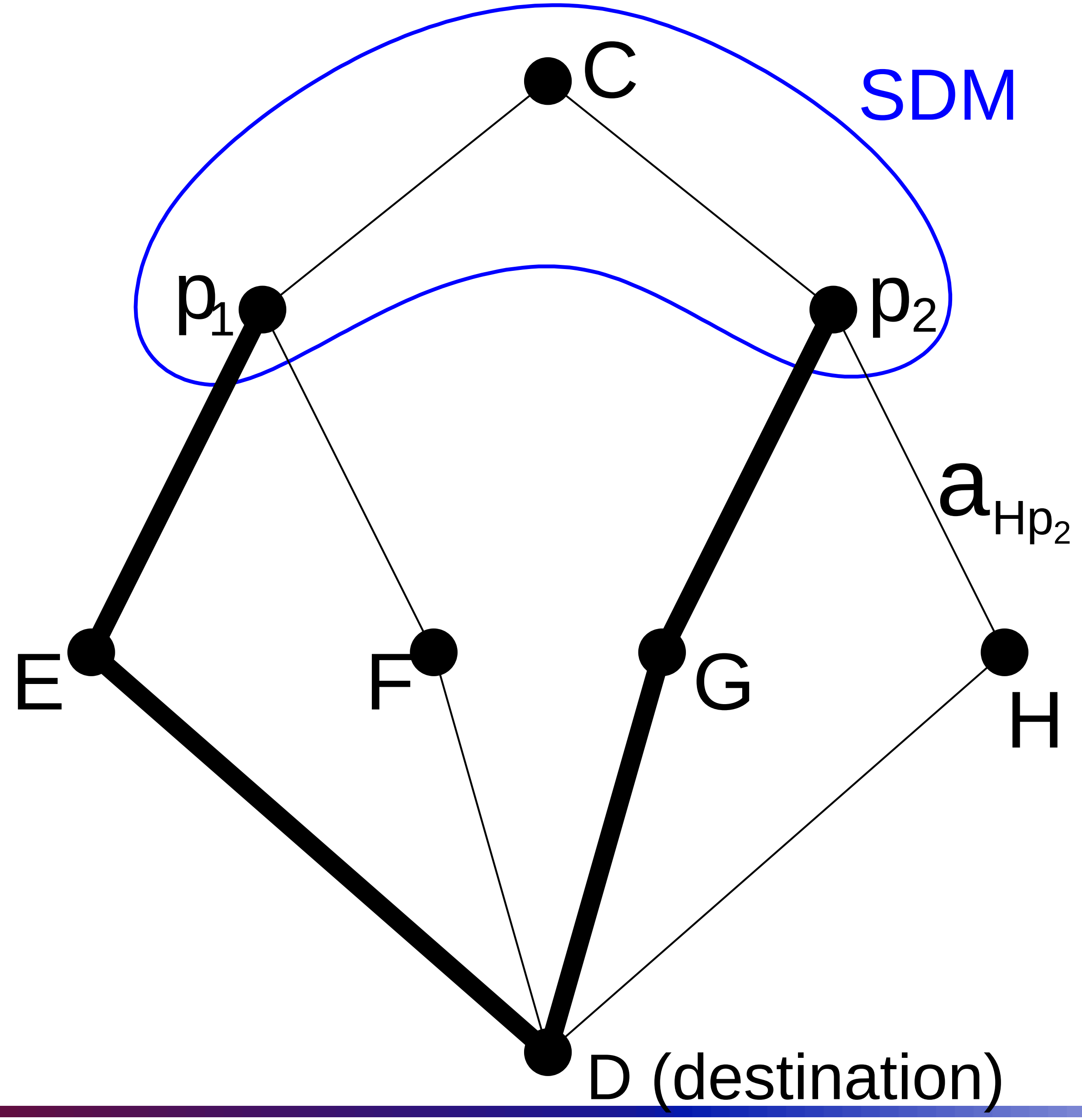
STRIP step 2 (a-b)



STRIP step 2 (c-d)



STRIP step 2 (e-f)



STRIP leakage

- Its better than SRIP
  - no-one learns any real distances
  - no-one learns relative distances
- but C does multiple computations
  - might infer something about R
  - C can learn partial ordering during convergence
- STRIP++
  - We can restrict information leakage
    - split information being sent along paths so that no-one sees metric sums
    - no leakage of any of values

Scalability

- there is a cost to secrecy
- increased communications overhead
  - SDM has $O(NK \log_2 n)$ communications overhead
    - C has N neighbours
    - metrics lie in the set $\{1, 2, \ldots, K\}$
    - probability of a mistake is $1/n$
  - request/response $O(NL \log K)$ communications overhead
    - average path length is L
- SRIP only need SDM
- STRIP needs both parts

Conclusion

- we can do stuff that I never imagined (until very recently)
- some of it is really cool

Future

- application to path-vector
- integration with security (authentication)

Bonus slides

OT - how it works

1-in-2 Oblivious Transfer

- Alice has a pair of bits (a_0, a_1), and Bob has β
- trapdoor permutation f
  - Given key k, can choose permutation pair (f_k, f_k^{-1})
  - Given f_k it is hard to find f_k^{-1}
  - Easy to choose random element from f_k's domain
- random Bit B_{f_k} is a poly.-time Boolean function
  - $B_{f_k} = 1$ for half of the objects in f_k's domain
    $B_{f_k} = 0$ for other half
  - no probabilistic polynomial time algorithm can make a guess for $B_{f_k}(x)$ that is correct with probability better than $1/2 + 1/\text{poly}(k)$

1-in-2 Oblivious Transfer

- A randomly chooses (f_k, f_k^{-1}), and tells f_k to B
- B randomly chooses x_0 and x_1 in f_k's domain, and computes $f_k(x_i)$
- B sends A the pair

$$(u,v) = \begin{cases} (f_k(x_0), x_1), & \text{if } \beta = 0 \\ (x_0, f_k(x_1)), & \text{if } \beta = 1 \end{cases}$$

- A computes $(c_0, c_1) = \left(B_{f_k}(f_k^{-1}(u), f_k^{-1}(v))\right)$
- A sets $d_i = a_i \text{ xor } c_i$ and sends (d_0, d_1) to B
- B computes $a_\beta = d_\beta \text{ xor } B_{f_k}(x_\beta)$

http://www.cs.ut.ee/~lipmaa/crypto/link/protocols/oblivious.php

Dining cryptographers

- N cryptographers are having dinner
- When it is time to pay the bill, the waiter tells them that someone has already paid
- the cryptographers are suspicious by nature (particularly Alice and Bob).
  - they suspect the NSA has paid
- not wanting to be compromised by such an association, they need to find out if someone at the table paid, or an external party such as the NSA
- how can they do so, without anyone revealing whether they paid or not?
  - of course, the waiter is sworn to secrecy

Secure Distributed Summation

Problem: N parties each have one value v_i and they want to compute the sum

$$V = \sum_{i=1}^{N} v_i$$

but they don't want any other party to learn their value.

SDS algorithm [6]

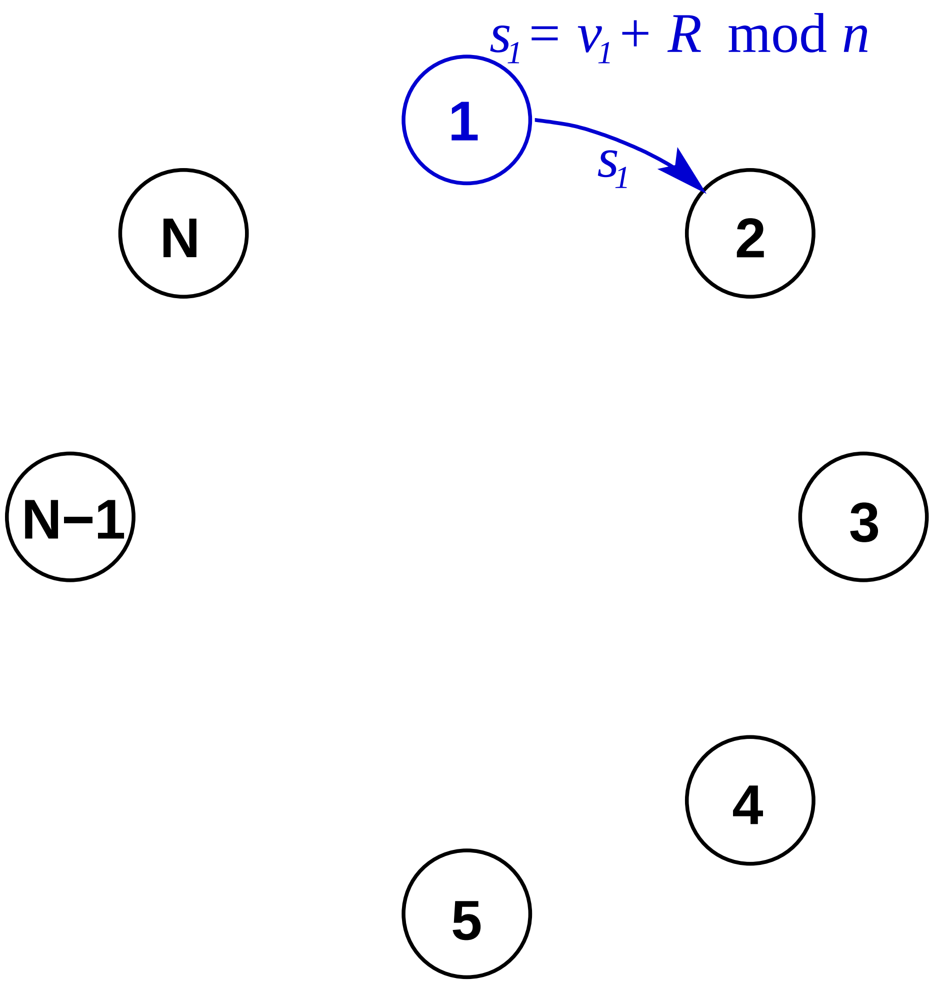
Assume the value $V \in [0,n]$ (for large n)

```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R mod n
party 1:  pass s_1 to party 2
for i=2 to N
  party i:  compute s_i = s_{i-1} + v_i mod n
  party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R mod n
```

Finally, party 1 has to share the result with the others.

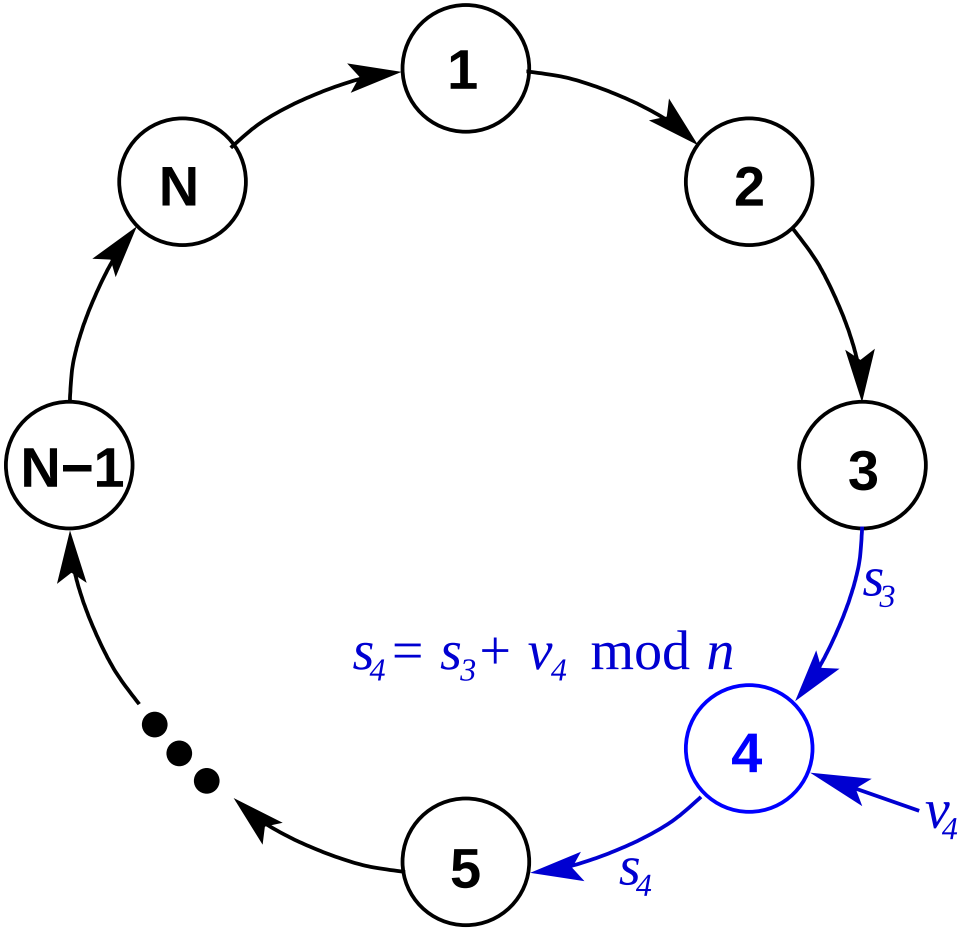
s_i will be uniformly randomly distributed over $[0,n]$ and so we learns nothing about any other parties values.

SDS algorithm



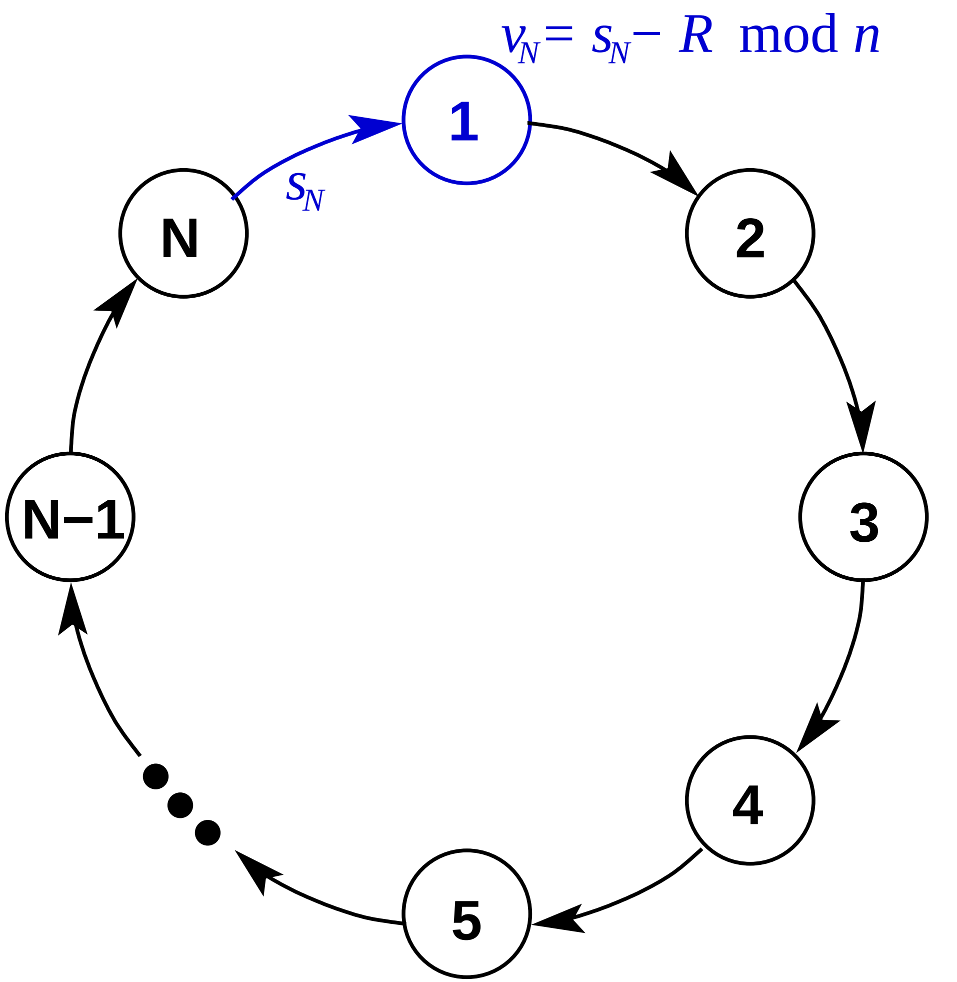
party 1: randomly generate $R \sim U(0,n)$
party 1: compute $s_1 = v_1 + R \mod n$
party 1: pass s_1 to party 2
for i=2 to N
   party i: compute $s_i = s_{i-1} + v_i \mod n$
   party i: pass s_i to party $i+1$
endfor
party 1: compute $v_N = s_N - R \mod n$

SDS algorithm




```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R  mod n
party 1:  pass s_1 to party 2
for i=2 to N
   party i:  compute s_i = s_{i-1} + v_i  mod n
   party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R  mod n
```

SDS algorithm




```
party 1:  randomly generate R ~ U(0,n)
party 1:  compute s_1 = v_1 + R  mod n
party 1:  pass s_1 to party 2
for i=2 to N
   party i:  compute s_i = s_{i-1} + v_i  mod n
   party i:  pass s_i to party i+1
endfor
party 1:  compute v_N = s_N - R  mod n
```

Applications

- dining cryptographers
  - v_i equals 1 if a diner paid, zero otherwise, $n = 1$, and $V \in \{0, 1\}$
- calculating the total traffic on the Internet
  - v_i is total per ISP
  - need some care to avoid double-counting
- Internet health (e.g. by accumulating certain statistics, e.g. packet drops)
  - e.g. v_i is packet loss percent at each ISP
  - use sum to compute (weighted) average
  - time series algorithms (either pre- or post-)
- Sketches

Collusion

- Assume party j and $j+2$ collude
  - They know at least s_j and s_{j+1}
  - $s_{j+1} - s_j \bmod n = v_j$
  - so they can learn the value of j
- Various methods of prevention, e.g.
  - divide v_i randomly into shares v_{im} such that

$$\sum_m v_{im} = v_i$$

  - sum over i in a different order for each m.

$$\sum_{i=1}^{N} v_{im} = V_m$$

  - sum V_m normally $V = \sum_m V_m$

SDP - how it works

(1) A and B agree on two numbers m and n

(2) A finds m random vectors $\mathbf{t}_i$ such that

$$\mathbf{a}_1 + \mathbf{a}_2 + \ldots + \mathbf{a}_m = \mathbf{a}$$

B finds m random numbers $r_1, r_2, \ldots, r_m$.

(3) for $i = 1$ to m

(3a) A sends B n different vectors:

$$\{\mathbf{a}_i^{(1)}, \mathbf{a}_i^{(2)}, \ldots, \mathbf{a}_i^{(n)}\}$$

where exactly one $\mathbf{a}_i^{(q)} = \mathbf{a}_i$, the other $n-1$ vectors are random

(3b) B computes $\mathbf{a}_i^{(j)} \cdot \mathbf{b} - r_i$

(3c) A uses 1-in-n OT to retrieve

$$v_i = \mathbf{a}_i^{(q)} \cdot \mathbf{b} - r_i = \mathbf{a}_i \cdot \mathbf{b} - r_i.$$

(4) B computes $V_b = \sum_{i=1}^{m} r_i$

(5) A computes

$$V_a = \sum_{i=1}^{m} v_i = \sum_{i=1}^{m} \mathbf{a}_i \cdot \mathbf{b} - r_i = \mathbf{a} \cdot \mathbf{b} - V_b.$$

References

[1] "Data-mining moratorium act of 2003." Introduced in Senate of the United States in January 2003. http://thomas.loc.gov/cgi-bin/query/z?c108:S.188:.

[2] A. M. Odlyzko, "Internet traffic growth: Sources and implications," in Optical Transmission Systems and Equipment for WDM Networking II (B. B. Dingel, W. Weiershausen, A. K. Dutta, and K.-I. Sato, eds.), vol. 5247, pp. 1–15, Proc. SPIE, 2003.

[3] A. Yao, "Protocols for secure computations," in Proc. of the 23th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 160–164, 1982.

[4] H. Lipmaa, "Oblivious transfer or private information retrieval." http://www.cs.ut.ee/~lipmaa/crypto/link/protocols/oblivious.php.

[5] B. Pinkas, "Oblivious transfer." http://www.pinkas.net/ot.html.

[6] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Zhu, "Tools for privacy preserving distributed data mining," SIGKDD Explorations, vol. 4, December 2002.

[7] Y. Lindell and B. Pinkas, "Privacy preserving data mining," Journal of Cryptology, vol. 15, no. 3, 2002.

[8] "Internet Activity, Australia." http://www.abs.gov.au/Ausstats/abs@.nsf/0/6445f12663006b83ca256a150079564d?OpenDocument, 2005.

[9] S. Muthukrishnan, "Data streams: Algorithms and applications," 2003. Manuscript based on invited talk from 14th SODA. Available from http://www.cs.rutgers.edu/~muthu/stream-1-1.ps.

[10] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," Proceedings of Latin American Theoretical Informatics (LATIN), pp. 29–38, 2004.

[11] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in Proc. of the Annual Computer Security Applications Conference (ACSAC '2001), (New Orleans, LA, USA), December 2001.