Sequential and distributed computation of routing paths across the internet

João Luís Sobrinho

September 5, 2025

1 Internet routing

The internet is an interconnected collection of about 75,000 networks, called Autonomous Systems (ASes). Two ASes are neighbors if they have one link in each direction between them, allowing them to exchange traffic. Every pair of neighbor ASes engages in a commercial relationship. For simplicity, just two types of relationships are considered: (1) in a *provider-customer* relationship, the customer pays the provider to transit its traffic to and from the rest of the internet; (2) in a *peer-peer* relationship, the two peers exchange traffic between themselves and their customers without charges. The customer-provider relationships create an hierarchy in the Internet. At the bottom of the hierarchy are ASes without customers, called *stubs*. All other ASes are *transit ASes* belonging to an Internet Service Provider (ISP), the business of which is to transit traffic. At the top of the hierarchy are ASes without providers, called *Tier-1s*. The majority of ASes are stubs, which include: (1) content provider and content distribution networks, such as those belonging to Google, Facebook, Amazon, Microsoft, Apple, Netflix, Akamai, and Cloudflare; (2) access networks, university networks, and enterprise networks. There are only a dozen or so Tier-1s, including AT&T, Lumen Technologies, Verizon Enterprise Solutions, Deutsche Telekom Global Carrier, Telxius, NTT communications, and Tata Communications. Wikipedia presents an updated list of Tier-1 ASes. The Center for Applied Internet Data Analysis (CAIDA) provides inferred topologies of the internet annotated with the type of commercial relationship between neighbor ASes (www.caida.org). For more information on inter-AS routing, the interested student can start from the CAIDA site and the following references: Gao and Rexford [2001], Gill et al. [2013], Mazloum et al. [2014], Kastanakis et al. [2023].

2 Routing paths

There are two important factors to understand internet routing. First, is how paths are characterized, how their characteristics are composed from those of its links, and how those characteristics determine a preference among paths. Second, is the constraint imposed on routing paths by the fact that data-packet forwarding depends exclusively on destinations: internet routing is not optimal-path routing. We consider these in turn.

2.1 Characterization of paths

A path is *valid* if, and only if, all intermediate ASes along the path are paid to transit traffic, either by their upstream or their downstream AS neighbors along the path, or by both. Valid paths have a *type*, which is one of customer, peer, or provider. A *customer* (*peer*, *provider*) path is a valid path that starts with a link from an AS to a customer (peer, provider) AS. A customer path is preferred to a peer path, which is preferred to a provider path. Among customer (peer, provider) paths, those of shorter length, measured by the number of links, are preferred.

The description above can be modeled precisely. Every link and path is associated with a *type-length* pair containing a type and a length. The special type-length $\bullet$ represents an invalid path. Types customer, peer, and provider, are represented by the integers 1, 2, and 3, respectively, while lengths are represented by nonnegative integers. Type-lengths are compared through the *lexicographic order*, denoted by $\preceq$. Specifically, for all type-lengths (α, n) and (α', n'),, other than $\bullet$,

$$(\alpha, n) \preceq (\alpha', n'), \text{ if } \alpha < \alpha' \vee (\alpha = \alpha' \wedge n \leq n').$$

From this comparison relation, a *selection operation* $\sqcap$ is defined such that for every nonempty set X of type-lengths, $\sqcap X$ is the most preferred of the type-lengths in X.

The type-length of a path is obtained from those of its constituent links through an *extension operation*, denoted by $\oplus$. Specifically, for all type-lengths (α, n) and (α', n'), other than $\bullet$,

$$(\alpha, n) \oplus (\alpha', n') = \begin{cases} (\alpha, n + n'), & \text{if } \alpha = 3 \vee \alpha' = 1; \\ \bullet, & \text{otherwise.} \end{cases}$$

Note that extension operation $\oplus$ is neither commutative, $(1,1) \oplus (3,1) = \bullet \neq (3,2) = (3,1) \oplus (1,1)$, nor associative, $((3,1) \oplus (1,1)) \oplus (3,1) = (3,3) \neq \bullet = (3,1) \oplus ((1,1) \oplus (3,1))$. The type-length of link uv is denoted by $tl[uv]$: it is $(1,1)$, $(2,1)$, and $(3,1)$, respectively, for a customer, peer, and provider link. The type-length of path uvP, obtained from the concatenation of link uv with path P, is denoted by $tl[uvP]$, and is computed from $tl[uv] \oplus tl[P]$.

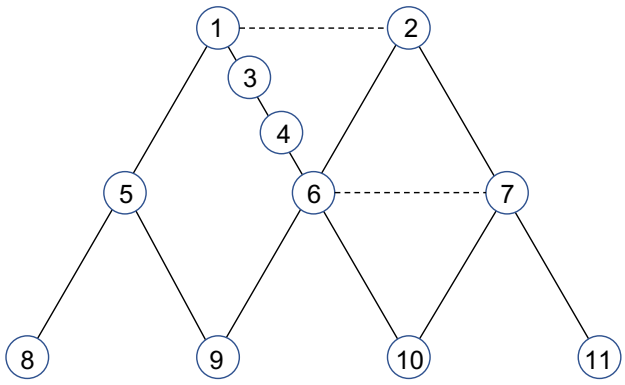

Figure 1: An internet. Solid lines join providers to customers, with providers drawn above customers. Dashed lines join peers.

Let us see some examples from the internet of Figure 1, where a provider is joined to a customer with a solid line, the provider drawn above the customer, while two peers are joined by a dashed line. AS 1 is a provider of both AS 3 and AS 5, and a peer of AS 2. Each line represents two links in opposite directions.

- The shortest path from AS 5 to AS 6 is $5-9-6$. However, that path is not valid because, both being providers of AS 9, neither AS 5 nor AS 6 pays AS 9 to transit traffic. Therefore, the type-length of path $5-9-6$ is $\bullet = (1,1) \oplus (3,1)$.
- Path $1-3-4-6$ is a customer path containing three links. Its type-length is $(1,3) = (1,1) \oplus ((1,1) \oplus (1,1))$. Path $1-2-6$ is a peer path. Its type-length is $(2,2) = (2,1) \oplus (1,1)$. Therefore, path $1-3-4-6$ is preferred to path $1-2-6$, expressed as $(1,3) \prec (2,2)$.
- Path $5-1-3-4-6$ is a provider path. Its type-length is $(3,4) = (3,1) \oplus (1,3)$. Path $5-1-2-6$ is also a provider path. Its type-length is $(3,3) = (3,1) \oplus (2,2)$. Therefore, path $5-1-2-6$ is preferred to path $5-1-3-4-6$, expressed by $(3,3) \prec (3,4)$.

2.2 Stable destination-based routing

Data-packet forwarding in the internet is *destination-based*. Every AS maps each destination to at least one neighbor to which all data-packets addressed to that destination are forwarded. The choices of forwarding neighbors over all ASes to a common destination are such that if the individual choice at any one node changes unilaterally, then data-packets traveling from that AS to the destination will not traverse a path with a preferable type-length. These forwarding neighbor choices lead to an equilibrium among type-lengths that we call *stable destination-based routing*, or

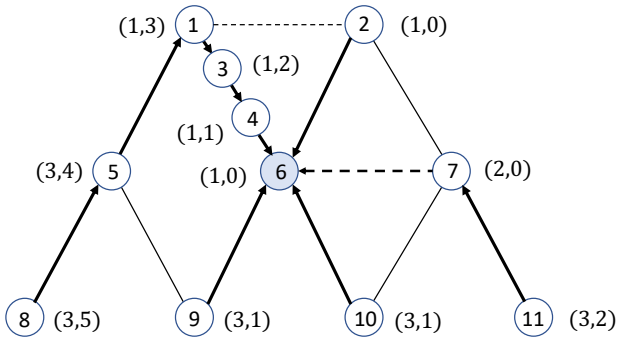

Figure 2: Stable routing to destination 6. Pairs represent type-lengths and thick lines represent links to forwarding neighbors.

stable routing, for short. Let N_u^+ denote the set of out-neighbors of AS u. A stable routing to t is a map E_t from ASes to type-lengths such that

Stability. $E_t(t) = (1,0)$, and for every other AS u,

$$E_t(u) = \sqcap\{tl[uv] \oplus E_t(v) \mid v \in N_u^+\}.$$

In-tree. For every u, there exists $u^+ \in N_u^+$ with $E_t(u) = tl[uu^+] \oplus E_t(u^+)$, such that the graph induced by all links of the form uu^+ is an in-tree rooted at t. Node u^+ is the forwarding neighbor at u to t.

The stability condition at node u is equivalent to the statement that $E_t(u) \preceq tl[uv] \oplus E_t(v)$, for all $v \in N_u^+$, and $E_t(u) = tl[uv] \oplus E_t(v)$, for some $v \in N_u^+$. The in-tree condition rules out forwarding loops and implies that $E_t(u)$ is the type-length of paths followed by data-packets from u to t.

Figure 2 shows the stable routing in the network of Figure 1 to destination 6. For instance, the stable routing type-length at AS 1 is $(1,3)$, corresponding to customer path $1-3-4-6$ of length three, while the stable routing type-length at AS 5 is $(3,4)$, corresponding to provider path $5-1-3-4-6$ of length four. Note that the stable routing paths are not optimal paths, in general. For instance, the optimal path from AS 5 to AS 6 is provider path $5-1-2-6$ with length three.

3 Sequential algorithm

You will be asked to design and implement algorithms that compute stable routing type-lengths and forwarding neighbors, and, using them, extract statistics concerning their types and lengths. The input to all algorithms includes an internet given as a file. Each line of the file describes a

link; it consists of a tail AS, a head AS, and the commercial relationship the tail has with the head. AS identifiers are integers between 0 and $2^{16} - 1 = 65535$. As said before, 1, 2, and 3 represent a link from provider to customer, from peer to peer, and from customer to provider, respectively. For example, in the list below, the first two lines imply that AS 4323 is a provider of AS 12122 and the last two lines imply that AS 29017 and AS 34309 are peers. It is recommended that the

4323	12122	1
12122	4323	3
7018	17228	1
17228	7018	3
29017	34309	2
34309	29017	2

input internet be represented in main memory with an adjacency list and that a total number of $2^{16} = 65536$ ASes be represented, even though some of them will not be part of the internet, having neither incoming nor outgoing links. Files containing internets of increasing size will be provided during the course.

You are asked the following:

- Design and implement algorithm STABLETYPELENGTH(*Net*,*t*) that receives as input internet *Net* and a destination *t* in that internet and computes the stable routing type-lengths from each ASes to destination *t*. The program wrapping the algorithm should have a mode where the name of the file containing *Net* and the destination *t* are read from the standard input (keyboard) and the results are written to the standard output (screen).

- Using STABLETYPELENGTH(*Net*,*t*), design and implement algorithm STABLEALL(*Net*) that returns statistics of stable routing type-lengths over all destinations. Concretely, we want to know the fraction of types that are customer, peer, provider, and invalid, and the Complementary Cumulative Distribution Function (CCDF) of lengths.

Students should start experimenting with "paper and pencil" on small networks how the requested algorithms should be designed. Some underlying theory will then be discussed in the fourth or fifth lecture. In general, it is easier to design an algorithm that computes stable routing type-lengths than an algorithm that computes optimal type-lengths from an AS to a destination. The interested student may also want to explore how we design an algorithm that computes optimal type-lengths.

4 Distributed algorithms

In the internet, the computation of stable routing type-lengths and forwarding neighbors is performed by distributed algorithms, called routing protocols. These protocols instantiate a separate routing computation per destination. Fix a destination t. AS t initiates the computation by advertising type-length $(1, 0)$ to all its in-neighbors by means of routing messages. Note that routing messages travel from head to tail of a link, in opposite direction to that of data-packets. Every node u maintains state variable stl_u that stores an estimated type-length to the destination. Node u keeps its in-neighbors updated with its current type-length estimate by advertising it to all of them by means of routing messages. The computation terminates when there are no more routing messages in transit across the links of the network. We will consider two routing protocols: the *simple stable routing protocol* and the *complete stable routing protocol*.

Simple. The only variable maintained at AS u to destination t is stl_u. Suppose that node u receives a routing message with type-length (α, n) from its out-neighbor v. Then, AS u: (i) extends the type-length of its link to v with (α, n), obtaining type-length $tl[uv] \oplus (\alpha, n)$; (ii) stores the obtained type-length in stl_u and advertises it to all its in-neighbors if, and only if, it is preferred to the previous type-length stored in stl_u.

Complete. In addition to stl_u, AS u maintains variable $stl\_tab_u[v]$, for every out-neighbor v of u, that stores the type-length to t last learned from v. At all times, stl_u equals the most preferred of the variables $stl\_tab_u[v]$, that is,

$$stl_u = \sqcap\{stl\_tab_u[v] \mid v \in N_u^+\}.$$

Suppose that node u receives routing message with type-length (α, n) from its out-neighbor v. Then, AS u: (i) extends the type-length of its link to v with (α, n), obtaining $tl[uv] \oplus (\alpha, n)$, and stores the extension in $stl\_tab_u[v]$; (ii) re-computes the preferred type-length estimate stl_u; (iii) advertises the new type-length estimate to all its in-neighbors if, and only if, it has changed.

One interesting difference between the simple and the complete routing protocols is that in the former, the type-lengths stored in variable stl_u can only decrease during an execution, whereas in the latter they can both decrease and increase.

You are not going to implement the simple and complete stable routing protocols with a separate agent per node, but rather you will build a *discrete-event simulator* for each protocol. The underlying assumptions of the simulator are: (i) routing messages are delivered across each link in First-In First-Out (FIFO) order; (ii) each routing message incurs a delay of at least one unit of

time plus a random value taken from a uniform distribution between zero and a parameter d; (iii) each routing message arrives at least 0.01 units of time after the last one. The appendix explains how to compute the arrival times of routing messages from their departure times to satisfy the above conditions. The operation of a discrete-event simulator revolves around a *calendar list* containing all events that are currently scheduled ordered by their time of occurrence. At each step of the simulation, the next scheduled event is removed from the calendar and processed according to the specific rules of the protocol. This processing may schedule subsequent events to be inserted in the calendar, in due order, for future processing.

You are asked the following:

- Implement SIMUSIMPLE(*Net*,*t*,*d*) that receives as input internet *Net*, a destination t, and a bound d on random delays, and computes the stable routing type-lengths from all ASes to destination t. The program wrapping the simulator should have a mode where the name of the file containing *Net*, the destination t, and bound d, are all read from the standard input (keyboard) and the results are written to the standard output (screen).

- Implement SIMUCOMPLETE(*Net*,*t*,*d*) that receives as input internet *Net*, a destination t, and a bound d on random delays, and computes the stable routing type-lengths from all ASes to destination t. The program wrapping the simulator should have a mode where the name of the file containing *Net*, the destination t, and bound d, are all read from the standard input (keyboard) and the results are written to the standard output (screen).

- Using SIMUCOMPLETE(*Net*,*t*,*d*), design and implement algorithm SIMUCOMPLETEALL(*Net*, *d*) that returns statistics of stable routing type-lengths over all destinations. Concretely, we want to know the fraction of types that are customer, peer, provider, and invalid, and the CCDF of lengths. Present as well the CCDF of the number of routing messages exchanged until the protocol terminates.

Students can start implementing the simulators right away, as these implementations are independent from the lectures. They should expect the size of the networks supported by the simulators to be significantly smaller than the size of the networks supported by the sequential algorithms.

5 Report and evaluation

Students work on the project in groups of two. Each group should design, implement, and test the algorithms requested. Each group will write a report with no more than five pages summarizing its findings. The report must include the following:

- A clear description of algorithm STABLETYPELENGTH(*Net*,*t*) in pseudo-code. Choose expressive names for the variables, and structure and comment the pseudo-code.
- A discussion of the complexity of algorithm STABLETYPELENGTH(*Net*,*t*).
- The statistics obtained by algorithms STABLEALL(*Net*) and SIMUCOMPLETEALL(*Net*, *d*) (you can choose the value of *d*, different from 0). Devise a few input networks to test your algorithms.
- Overall conclusions about the subject of this project.

The following aspects are considered in the evaluation:

- I will start by reading your report, which is the means for you to communicate your ideas with others. Organize your report in sections; present high-level, but precise descriptions of your algorithms, highlighting their most subtle steps, if any; draw concise, but unambiguous conclusions.
- If the report is readable, I will run some tests on your programs.
- I will have a discussion with each group, at which time we test your programs and debate your choices. The discussion occurs at the end of the course.
- A mark with be given individually to each member of the group.

I will keep the option of asking for some small, extra requirements on the project. Thus, please stay in tune. You are welcome to share ideas with your colleagues, but the design and implementation of the algorithms and the text of the report must absolutely be your own.

The code and the report should be sent in a .zip file to my email address with subject "[ARA] <group number>.zip',' where <group number> is your group number. The deadline is Friday, October 10 at 23:59. The discussions will occur during the week starting October 20, in the lab sessions.

Bibliography

L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, December 2001.

P. Gill, M. Schapira, and S. Goldberg. A survey of interdomain routing policies. *ACM SIGCOMM Computer Communications Review*, 44(1):28–34, December 2013.

S. Kastanakis, V. Giotsas, I. Livadariu, and N. Suri. Replication: 20 years of inferring interdomain routing policies. In *Proc. ACM Internet Measurement Conference*, pages 16–29, 2023.

R. Mazloum, M.-O. Buob, J. Augè, B. Baynat, D. Rossi, and T. Friedman. Violation of interdomain routing assumptions. In *Proc. International Conference on Passive and Active Network Measurement*, pages 173–182, 2014.

A Implementing a FIFO queue with random delays

Consider a sequence of messages sent along a FIFO channel running from a node v to a node u. The departure time of message i at v is denoted by d_i and the arrival time of this message at u is denoted by a_i, $i \geq 1$. If the channel were empty when message i is sent, then the delay of this message would be $d_i = 1 + D_i$, where D_i is a uniformly distributed random variable in the interval $[0, d]$. However, the channel typically holds earlier messages waiting to be delivered. Taking queuing into account, the arrival time of message i is given by

$$\begin{aligned} a_1 &= t_1 + d_1; \\ a_{i+1} &= \max(a_i + 0.01, t_{i+1} + d_{i+1}), \quad \text{for } i > 1. \end{aligned}$$