Urbit Networking: Ad Fontes

Edward Amsden ~ritpub-sipsyl
Zorp Corp

Abstract

Contents

1 Urbit Networking: Ad Fontes

1.1 In Principio

On Mars, SIN is taken to an extreme. Logically, Urbit is a single broadcast network - a single big Ethernet wire. Everyone sees everyone else's packets. You don't send a packet. You "release" it. As a matter of practical optimization, of course, routing is

Manuscript submitted for review.

Address author correspondence to ~ritpub-sipsyl.
(Edited from the original screed ~ritpub-sipsyl, ~2022..)

> necessary, but it is opaque to the receiver. If there is a routing envelope, it is stripped before processing.
>
> An easy way to see this is to compare an Urbit packet, or card, to an IP packet. An IP packet has a header, which is visible both to the routing infrastructure and the application stack. The header, of course, contains the infamous source address. Urbit removes not just the source address, but also the rest of the header. An Urbit card is all payload - it has no region seen both by receiver and router.

`~sorreg-namtyv`, ~2010..

> In the classical stack, a basic result of protocol design is that you can't have exactly-once message delivery . To put it in the terms we in Section 3.2: you can't build a bus on a network. With permanent networking between solid-state interpreters, this feature is straightforward. Why? Because two uniformly persistent nodes can maintain a permanent session

> Arvo defines a typed, global, referentially transparent namespace with the Ames network identity (page 34) at the root of the path. User-level code has an extended Nock operator that dereferences this namespace and blocks until results are available. So the Hoon programmer can use any data in the Urbit universe as a typed constant.

`~sorreg-namtyv` et al., ~2016..

1.2 Nunc

Today there is Ames. Ames provides a *command* protocol with exactly-once semantics over the wire. Urbit ships can *poke* other ships and expect eventually once, and only once, action on the poke provided that at some point in the future both

hosts are able to communicate. Urbit ships can also *watch* a path (wire) on another ship and receive updates, provided the subscription is not kicked. Both commands and updates must be acknowledged as a core requirement of the protocol.

What does not exist is a viable implementation of the typed, global, referentially transparent namespace. Neither the interface for application writing, nor the implementation of the stack beneath it, support programming by binding paths in a namespace and apprising other ships of such bindings or permitting them to request such bindings.

This document is in aid of the effort to implement such an interface and networking stack.

1.3 Observations

> **As a matter of practical optimization, of course, routing is necessary, but it is opaque to the receiver.**

`~sorreg-namtyv`, ~2010..

This is a massively important observation. Arvo should not "do routing." It should not be aware of sources or destinations. The runtime may route. Hoon code kept up to date by Arvo for the runtime's use may route. Agents may route. Non-Urbit or quasi-Urbit entities on the network may route. Arvo, a kelvin-versioned artifact which must not be subject to the uncertainty of external technical development, must not route.

1.4 The stack

From `~rovnys-ricfer`:

- PKI
- routing and peer discovery
- transport layer (including packetization)
- scry resolution layer
- frontier discovery layer

- command layer

The PKI can be treated for now as a solved problem. Routing and peer discovery must have presently viable implementations, but nothing which is being kelvined should constrain future, more Urbit-maximalist implementations from creation. Likewise the transport layer. Urbit needs to function both now, over various Ethernets, and millenia from now, over the Third and Most Glorious Quasiluminal Marterran Subspace Transponsive. Packetization is not something that should be kelvin versioned. None of these problems cry out for an immediate solution, though packetization ought not to churn Arvo's event loop.

1.4.1 Scry resolution

The scry resolution layer is where the rubber meets the road. How are paths bound by remote ships resolved to data? The only possible way is for some ship (initially the binding ship, and later possibly other ships and even non-urbit transponders) to communicate the binding. Since bindings are irrevocable, we call a particular binding an "oath" and we say that a ship will "avow" the oath by taking action to inform others of it.

A ship has no possible way to force its runtime to do anything. Everything depends on the runtime's operations on a given and current state of Arvo. (More of the runtime should be written in Hoon, it should be said.) But a ship can advise or insist that the runtime take some action. Oaths are immutable but may be avowed many times.

1.4.2 Frontier discovery

The frontier is, at this point in time, the limit of what is knowable. In general we do not know all that is knowable. Many ships have bound and even avowed oaths which we have not yet heard of, and may never hear of. But we would like to be able to reliably discover what oaths our peers will avow to us.

Unfortunately, ***there is no way to do this which is optimal for all applications.*** The search for such a mechanism is

at the root of much of the conceptual and design difficulty for Urbit's networking layer.

We want low-latency, push-based subscriptions. We want low load on large publishers. There is a fundamental tension here.

The correct approach is to provide an interface for applications to implement different strategies in this regard. No networked application can demand absolute upper bounds on delivery latency *and* absolute reliability.

For example: a chat application might work by eagerly broadcasting avowals of new chat messages, and then beaconing out reavowals in the interim between new chat messages on some configurable but acceptable latency interval. Ten, fifteen, or even thirty seconds of time is not a steep penalty to pay for a dropped packet in a chat application. Of course, this penalty is only paid if the entire message is dropped. If we become aware of a message but we are missing some packets for it, we can always request (see below) an immediate re-send.

A media streaming application, whether for one-to-many broadcast or two-way or many-way communication, may never reavow a binding, since a binding which is known late is no longer relevant.

However, there should of course be a way to request a path from a ship. Thinking of this in terms of "which path exactly must I request?" is incorrect. The information being communicated is "I am interested in a (the latest?) path of this form", which information ought to be avowed, as an oath. This permits replication and caching of interest information, and application-specific but still not onerous handling of the problem of a dead or silent interest.

1.4.3 Command layer

The Ames command layer is, from the view of this system, an application, if a very general and ubiquitous one. A poke takes the form of an oath that you wish to poke another ship with some path and data. The command-layer application insists on re-avowing pokes on a decaying schedule until they are acknowledged. Acknowledgement is, yet again, an oath.

1.5 Routing and discovery, revisited

Alright then. Urbit is conceptually a broadcast network, but we necessarily optimize it by routing. So on what basis to we route, and to where?

Routing must be a decision taken on registration of interest. By what policy we determine interest may vary, but it should be informed primarily by avowals of interest by the destination. An exception is commands which should be routed unless *disinterest* is avowed.

The public Internet is not a broadcast network, and we cannot yet communicate over a global network where Urbit has its own ethertype and its own routers. Thus we must be able to select a set of public IPs to send avowals (our own or others) to.

For this reason, galaxies already live in DNS. Commands and registration of interest should by default be routed to sponsoring galaxies and be shared with other galaxies. Galaxies learn IPs of their sponsored stars by receiving such communication. Perhaps the registeration of interest is re-avowed on a heartbeat. Similarly, stars learn IPs of their planets by receiving registrations of interest. Any retransmitted avowal can have several lanes attached in the packet header (stripped before offering to Arvo) by which the avowing ship can be reached. Commands can be routed to the IP of the commanded ship, if known, its sponsor's IP, if known, or its sponsor's sponsor's IP. More general oaths are checked against registered interest (which may be expired if not re-avowed) and sent along known lanes to interested ships.

(In an Urbit maximalist future, BGP-type protocols between Urbit-aware routers will likely permit link-shaped routing of such avowals as well as sponsorship-shaped routing, which can reasonably exist now.)

Local networks, by contrast, generally permit and even depend on broadcast behavior. The Address Resolution Protocol is fundamental to the operation of local IP subnets over lower-layer networks. A packet is broadcast which requests a MAC address for an IP, and provides a MAC address to reply to. At higher layers this is replicated. Peer discovery for

local networks at the OS layer (DHCP, zeroconf networking), shared-hardware layer (printers and screens) and application layer (media) rely on on broadcast announcements which receive addressed replies. Commands and interest registrations could be broadcast on local networks to discover peers either directly, or via Urbit-aware routers.

References

~ritpub-sipsyl, Edward Amsden (2022) "Urbit Networking: Ad Fontes". URL: https://gist.github.com/eamsden/653c6055e8de10a291313f5ab37bf9e0 (visited on ~2024.5.15).

~sorreg-namtyv, Curtis Yarvin (2010) "Urbit: functional programming from scratch". URL: http://moronlab.blogspot.com/2010/01/urbit-functional-programming-from.html (visited on ~2024.1.25).

~sorreg-namtyv, Curtis Yarvin et al. (2016). *Urbit: A Solid-State Interpreter*. Whitepaper. Tlon Corporation. URL: https://media.urbit.org/whitepaper.pdf (visited on ~2024.1.25).