# An Axiomatic Basis for Communication

Martin Karsten
University of Waterloo
mkarsten@uwaterloo.ca

S. Keshav
University of Waterloo
keshav@uwaterloo.ca

Sanjiva Prasad
IIT Delhi
sanjiva@cse.iitd.ac.in

## ABSTRACT

The de-facto service architecture of today's communication networks lacks a well-defined and coherent theoretical foundation. With layering as the only means for functional abstraction, the diversity of current technologies cannot be expressed consistently and analyzed properly. In this paper, we present an axiomatic formulation of fundamental mechanisms in communication networks. In particular, we reconcile the existing but somewhat fuzzy concepts of *naming* and *addressing* and present a consistent set of primitives that are sufficient to compose communication services. The long-term goal of this exercise is to better document, verify, evaluate, and eventually implement network services.

## 1 INTRODUCTION

Traditionally, the Internet is modelled as a graph, where each node implements a set of protocol layers and each edge corresponds to a physical communication link. Unfortunately, when compared to the actual Internet, this model falls far short. In the traditional model, nodes are addressed by one or more static IP addresses. End systems implement a simple five-layer stack, with applications using a transport layer to access IP, which is layered on the data link and physical layers. Packet forwarding decisions are made purely on the basis of IP 'routing' tables. Moreover, a protocol layer at any node only inspects packet headers associated with that layer, obeying strict layering rules in dealing with other layers. In reality:

- DHCP, anycast, multicast, NAT, mobile IP and IP tunnelling break the static association between a node and its IP address.
- Nodes implement many more layers, including IP-in-IP and VLAN tunnels, P2P overlays, and shims, such as MPLS.
- Forwarding decisions are made not only by IP routers, but also by VLANs, MPLS routers, NAT boxes, firewalls, and mesh routing nodes.
- Middleboxes and cross-layered nodes such as NATs, firewalls, and load balancers violate layering.

In face of these significant extensions to the classical model, understanding the topology of the Internet in terms of its connectivity has become a daunting task. It has become difficult to even define elementary concepts such as a neighbour and peer relationships, let alone the more complex processes of forwarding and routing. Further, there is not even a common and well-defined language for fundamental networking concepts, with terms such as 'name', 'address', or 'port' being the subject of seemingly endless debate.

Yet, surprisingly, the system still works! Most users, most of the time, are able to use the Internet. What lies behind the unreasonable effectiveness of the Internet? We postulate that there are a set of underlying principles that are obeyed by extensions to the traditional model, no matter how ad hoc, which preserve connectivity. However, these principles have rarely been systematically studied (with [3, 4] being notable exceptions).

Our research goal, over the long term, is to axiomatically specify basic Internet concepts that allow us to construct (a) a theoretically sound framework to express architectural invariants–such as the deliverability of messages–even in the presence of network dynamism, middleboxes, and a variety of compositions of different protocols, and (b) an expressive pseudo-language in which to rapidly implement a variety of packet forwarding schemes. Therefore, the concepts, and the pseudo-language derived from them, serve not only to clarify the essential architecture of the Internet, but also provide a bridge between formal proofs on node reachability using a particular forwarding scheme, and a practical implementation of that scheme. Our goals are inspired by Hoare's axiomatic basis for programming [6].

In this paper, we take a first step in this direction by presenting an axiomatic framework of communication concepts and pseudo-language primitives derived from these concepts. We sketch how the framework can be formalized, but we do not discuss the implementation of the pseudo-language primitives. However, it will become clear that the primitives can be implemented in any reasonable packet forwarding engine.

To keep the problem tractable, we propose to split overall communication functionality into two broad areas: one area is concerned with *connectivity*, i.e. naming, addressing, forwarding, and routing. The second is the set of mechanisms to provide additional functionality related to communication quality and performance. This includes medium access control, reliability, flow control, congestion control, security, among others, and is not yet explicitly considered in our work.

Our work draws from, and is related to, a handful of other attempts to bring clarity to Internet architecture. Clark's seminal paper [3] succinctly laid out the design principles of the classical Internet, but does not provide a basis for formal reasoning about its properties. Recently, Griffin has used formal semantics to model routing [4] and Loo et al. have used a declarative approach to describe routing protocols [8]. Our work is directly related to past work in the area of naming and addressing indirection. This has been considered both in existing technology standards, such as IP Multicast, IPv6, or Mobile IP, as well as research proposals [2, 5, 9, 12]. Similar to our work, these proposals blur the traditional distinction between naming and addressing, and also consider innovative packet forwarding mechanisms. However, to our knowledge, these past proposals are essentially ad hoc, without a consistent set of underlying formal principles. In contrast, we suggest an axiomatic formulation of communication principles and thereby present a first attempt at building a complete formal basis for reasoning about communication systems.

# 2 Conceptual Framework

## 2.1 Naming and Binding

Naming and binding in computer systems is relatively well understood. Before introducing a new set of definitions, we first review and summarize some fundamental concepts from the seminal paper by Saltzer [11] with a few minor modifications, as noted.

- An *object* is a software or hardware structure in a computer system.
- A *name* is a regular expression that is used to refer to a set of objects. This is an extension from the original definition [11], which only refers to one string and one object. We use regular expressions to allow for wildcard matching and refer to a set of objects because of broadcast, anycast, and multicast communication styles.
- The original term *binding* [11] is used both as a noun, describing the existence of a mapping from a name to a set of objects, as well as a verb, referring to *choosing* the appropriate objects for a name. To avoid confusion, we use the term *mapping* when referring to the noun. In a traditional naming system, the single object can be accessed through a "lower-level" name [11], which is often called "address".
- A *context* is a set of mappings. A name is always interpreted relative to some context. To know the "lower-level" name associated with a name, one needs to also know which set of mappings is being referred to, because multiple contexts may provide different mappings for the same name.
- The *resolution* mechanism locates the appropriate mapping for a name in a particular context. This allows for access to the object through the corresponding "lower-level" name. The original definition is "locating the object" [11], which is identical to locating the mapping and accessing the object.

With these definitions, it is possible to develop the description of a basic naming system. However, this set of definitions stops at the concept of a "lower-level" name and simply assumes that it can be used to access a certain object. In a distributed system, however, the communication necessary to access a certain object is non-trivial and greatly influences the overall system behaviour.

## 2.2 Communication Concepts

Similar to Saltzer's usage of a "lower-level" name as primitive, we assume that certain objects can *directly communicate* with each other, without giving a formal definition of "direct communication". Direct communication is facilitated either by shared memory or takes place between low-level network entities that can directly exchange information via a physical medium, such as cable, radio, or fiber, for example in a local area network such as Ethernet. Based on this premise, we introduce the following definitions:

- We define a *network processing object (NPO)* as an object that can directly communicate with other NPOs. An NPO is an abstraction of a traditional protocol layer instance. Like any object, an NPO can be referred to by one or multiple names.
- An NPO may have a set of mappings associated with it, which is then called its *context state*. The set of mappings comprising the context state may contain wildcards. An example of a context state is a routing (or forwarding) table.
- NPOs that can directly communicate with one another are termed *neighbours*. An NPO can directly communicate with each of its neighbours using the neighbour's name. Examples of neighbouring NPOs are the TCP and IP NPOs on the same machine, or two MAC-layer NPOs on the same shared medium.
- The unit of communication is a *message*, which contains control information in the *header* and arbitrary data in the *payload*. The header might be explicit, as in a traditional packet header, or implicit, for example the time slot within a TDM frame during which a message is transmitted.

- A name that is encoded in the header of a message is termed an *address*. The message header contains, among other control information, a *stack* of addresses. The top-most address on the stack is the *destination address*.
- We define *forwarding* as an extension of direct communication, where NPOs repeatedly pass on a message to a set of neighbours, such that the message eventually arrives at a set of remote NPOs. In this sense, forwarding is the transitive relation of direct communication, necessary because not all NPOs are each others' neighbours.
- The original definition of *resolution* [11] needs to be generalized in that the result is not only a "lower-level" name, but includes further information to forward a message towards the set of NPOs referred to by a name.

Note that we have a particularly simple definition of an address: it is just a name that happens to be in a packet header and is therefore used to make a forwarding decision. This operational approach to defining an address bypasses myriad conceptual difficulties of other approaches. One immediate conclusion from this approach is that a name only needs to have local (per-NPO) syntax, while each address format must be standardized between NPOs. Note also that we explicitly describe each message as having a stack of addresses. When reading from and writing to the stack of addresses, multiple addresses may be transformed into one local name and vice versa. This allows us to model non-layered (or layer-violating) NPOs, such as middleboxes. For example, NAT operates on five address fields that internally are considered a single name.

## 2.3 Communication Operations

We define the local context state as the set of mappings from a name to a set of tuples of NPO and name as {<name → {<NPO, name>}>}.
The generic forwarding algorithm can then be described with the following pseudo-code using the primitives *send*, *receive*, *copy*, *push*, *pop*, *lookup*:

```
message msg = receive();
name n = pop(msg);
{<NPO, name>} S = lookup(n);
for each <NPO, name> s_i in S
  outmsg = copy(msg);
  push(outmsg, s_i.name);
  send(s_i.NPO, outmsg);
endfor
```

The `push` and `pop` primitives are specific to an NPO class and transform between a prefix of the address stack and a local name. Note that the above processing steps cover both ingress and egress processing for each NPO. Also, an NPO typically provides a *default mapping* which is used for all those names that do not have an explicit mapping in the context state. For example, in case of IP routing, this is called *default routing entry*.

The concepts and primitives introduced so far allow for the description of static communication scenarios, where forwarding tables and topologies do not change over time, and where local context state is sufficient to determine the neighbouring NPOs to whom a message should be forwarded. As an example, consider an IP network with pre-configured routing tables, running over Ethernet with all ARP lookups also pre-configured in the ARP cache.

## 2.4 Structure Concepts

The following definitions extend the basic communication concepts and allow to describe network structure at the familiar level of nodes and links.

- The NPO that inserts an address into a message header (by a `push` operation) along with those NPOs that *potentially* resolve the same address (using `lookup`) or remove it (`pop`) are termed *peers*.
- The communication association between a peer that writes a destination address into a message and a set of corresponding peers that receive the message and logically remove the destination address from the message (so that it it is no longer used for making a forwarding decision) is termed *link*. The sequence of peers forming a link is termed *path*.

Note that links are between peers. In contrast, neighbouring NPOs communicate via direct communication. Links are similar to ISO protocol interfaces, whereas direct communication refers to service interfaces. For example, an IP sender, IP router, and IP destination are peers, but not neighbours. A pair of connected Ethernet NICs can be considered as both neighbours and peers.

Using the concepts introduced so far, it is possible to describe data path mechanisms of a communication network. For example, we can talk of a link provided by two TCP NPOs that is established by a three way handshake. Similarly, a transient HTTP link exists between a browser client and a web server for the duration of the TCP link between them. An HTTP load balancer that examines the HTTP header would be a peer of the browser, and it would also be a peer to the web server. In this sense, the load balancer is a forwarding engine, on par with an IP router.

If suitable context state exists in all NPOs along a path, the message state necessary for forwarding a message to a set of remote NPOs can be reduced to a single name. Then, forwarding can be regarded as *binding* the name to

the set of destination NPOs. Based on this understanding, the following definitions provide concepts for structural properties.

- A set of peers that forward messages with the same destination address to the same set of NPOs provide *consistent* binding for this name.
- A *scope* for a set of names is a set of peers that provide consistent binding for each of these names. For each name in a scope, there is a unique sink tree leading to the NPO holding the corresponding mapping in its local context state. The full tree can be regarded as a distributed mapping.
- There can be special names in a scope, for example *broadcast* referring to all peers in a scope.
- Mechanisms and algorithms used to achieve consistency in a scope are collectively termed *routing*.

We assume that each individual NPO forms a natural scope for all names covered by its context state. Distributed control state such as distributed IP routing state can be described as a set of collaborating NPOs that form a scope for a set of names.

### 2.5 Distributed Resolution

The lookup primitive in Section 2.3 is defined on an individual NPO's context state and as such, inherently bound to a single object. A *context* as defined by Saltzer [11] is an abstract concept and not bound to any particular resource. Likewise, *resolution* is also not defined as local or distributed. While we assume that Saltzer implicitly refers to a local system only, the concepts work well in a distributed system.

Distributed resolution consists of forwarding a resolution request within the scope of the name to an NPO that has a mapping for this name and sending back the appropriate response. In a sense, the forwarding part is very similar to the definition of *closure* in Saltzer's work [11], which is defined as "the mechanism that connects an object wishing to resolve a name to a particular context".

For most scenarios, the definition of consistent binding nicely carries over to distributed resolution and resolution can be considered as binding a name to the appropriate object in a particular (distributed) context. The one exception is given by an anycast name that maps to multiple objects, but only a subset (typically one) of them is needed to successfully complete the task. In this case, consistency only applies to the set of eligible objects. In fact, caching in naming systems, being a special variant of replication, can be considered as anycast where the requested name can be bound to any of the available replicated objects. We note that implementations of anycast either rely on binding to some form of rendezvous point or employ a simplistic algorithm that does not change the original binding to a specific object, but shortcuts the resolution whenever possible, if a cached replica is found.

### 2.6 Control Operations

In this section, we introduce basic primitives that facilitate the interaction between control and communication operations. For simplicity and clarity, we do not model the algorithmic part of a control regime, for example distributed routing. Also, we ignore any access control that is necessary to validate control operations in reality.

Control operations are either triggered by special control messages (e.g. virtual circuit setup or routing), or implicitly depending on the message header (e.g. NAT setup). The corresponding details are beyond the scope of this paper. We only sketch the basic primitives that we envision to model control operations:

- `update(name, {<NPO, name>})`
  This primitive is used to add, remove, or update a mapping in context state.
- `create(name, op, {<NPO, name>})`
  This primitive creates and returns a control message containing the given arguments. The control message can be transmitted using the available forwarding primitives. The `LOOKUP` and `UPDATE` opcodes trigger the corresponding lookup or update operations at the destination NPO. The `INFO` opcode is used to communicate context state information to neighbours, for example routing information or name resolution replies.
- `control(name, op, {<NPO, name>})`
  This primitive represents the main entry point for control operations. For explicit control messages, the message content (cf. `create`) is passed and interpreted. For implicit control triggers, only the `name` is being used to determine the appropriate control operations. The algorithmic part of any control activity is also abstractly represented by this primitive. A detailed analysis of control operations is the subject of an ongoing study and will be presented in later work.

We use NAT as an example scenario between three nodes, as sketched in Figure 1, to illustrate the operation of and interaction between communication and control primitives. We ignore the ARP and outgoing Ethernet details to keep the example small and only show the processing inside the NAT node. Further, the execution of `pop` and `send` primitives is omitted from the example, since they are obvious. We use `UDP` as the local name of the UDP NPO instance at each node (aka "protocol number") and `IP` as the corresponding name for the IP

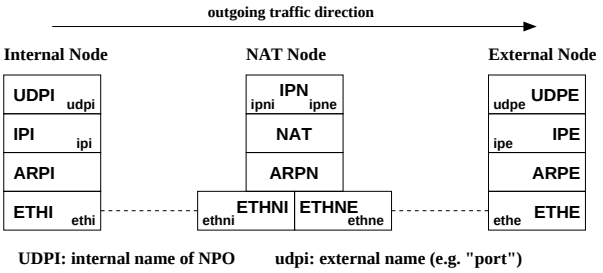


**Figure 1**: NAT Example Setup

instance. The address stack top to bottom is shown from left to right and we assume the convention that source addresses are pushed before destination addresses. We show the creation of a new UDP mapping:

```
ETHNI: lookup([ethni,ethi,IP])
NAT: control([ipe,ipi,UDP,udpe,udpi],
  NULL, <NULL, NULL>)
NAT: update([ipe,ipi,UDP,udpe,udpi],
  <IPN, [ipe,ipne,UDP,udpe,udpX]>)
NAT: update([ipne,ipe,UDP,udpX,udpe],
  <IPN, [ipi,ipe,UDP,udpi,udpe]>)
NAT: lookup([ipe,ipi,UDP,udpe,udpi])
NAT: push([ipe,ipne,UDP,udpe,udpX])
IPN: lookup([ipe,ipne,UDP])
IPN: push([ipe,ipe,ipne,UDP])
ARPN: lookup(ipe)
ARPN: push([ethe,ethne,IP])
ETHNE: ...
```

The Ethernet NPO at the incoming NIC performs a lookup for the destination address and the protocol field to determine whether the Ethernet frame should be received by this station and if yes, where to forward it to. The NAT NPO detects that the packet is sent from a non-local to an external address, but only finds the default mapping for this case, which invokes control processing. First, a free local UDP port `udpX` is determined, which is then used to create incoming and outgoing NAT context state. Afterwards, communication operations continue: The packet header is modified and the packet leaves via IP, ARP, and Ethernet NPOs.

The routing and forwarding of control requests and replies can be accomplished using available communication and control primitives. For example, the routing of reply messages, e.g. for resolution requests, is comparable to a NAT NPO that creates forwarding state for an outgoing messages and automatically routes the corresponding incoming messages.

# 3 FORMALIZATION

Formalization of the concepts introduced in the previous section provides a basis for rigorous analysis, validation and even formal verification of protocol design. We wish to operate at suitable levels of abstraction, and support modular analysis and refinement of specifications and formalizations. In this section, we sketch how a formal basis for our framework can be defined, and how formal analyses can be carried out.

## 3.1 Correctness Specifications

Consider the canonical requirement of *deliverability* of messages. This is easily specified as an *inductive* property of *a message msg at a given NPO np being deliverable to its destination NPOs*: (1) A message already at its destination NPO is deliverable. (2) A message *msg* at a particular NPO *np* is deliverable if *from each neighbour* $np_i$ in mapping $m$, obtained from looking up the destination name, message $msg'_i$, constructed as specified by $m$, is deliverable. This is formalizable in logic as a predicate *msg@np deliverable* using inductive inference rules. Inference rules with no assumptions are called *axioms*. Predicates defined in such a manner are amenable to inductive proof techniques, with automated theorem prover support.

## 3.2 Operational semantics

We formalize the operational semantics of constructing and deconstructing messages, and manipulating context state as an *abstract machine* in the style of [7] running at each NPO. Configurations of the abstract machine are triples $\langle S|cs|p\rangle$ consisting of (i) a stack of values manipulated by the NPO (names, messages, mappings,..), (ii) context state, and (iii) sequence of primitive operations to execute. This formalization has a well-understood theory [10] and mechanizable reasoning apparatus, which permits the use of algebraic analysis techniques.

The low-level transition relation $\longrightarrow$ describes the change in configuration. Arguments to the operations are implicitly specified; they are at the top of the stack in the "pre" configuration, and in the "post" configuration, the results of the operation are at the top of the stack. In each rule, the first primitive operation of the third component is executed; the remaining operations are subsequently performed from the "post" state onwards.

$$\langle (n_1 n_2 ..., d) ... | cs | \texttt{pop}; p' \rangle \longrightarrow \langle \{n_1, (n_2 ..., d)\} ... | cs | p' \rangle$$
$$\langle \{(n_2 ..., d), x\} ... | cs | \texttt{push}; p' \rangle \longrightarrow \langle (x n_2 ..., d) ... | cs | p' \rangle$$
$$\langle n ... | cs | \texttt{lookup}; p' \rangle \longrightarrow \langle m ... | cs | p' \rangle$$
provided $cs$ associates $m$ to name $n$
$$\langle \{n, m\} ... | cs | \texttt{update}; p' \rangle \longrightarrow \langle ... | cs[n \mapsto m] | p' \rangle$$
$$\langle \{n, m\} ... | cs | \texttt{create}(o); p' \rangle \longrightarrow \langle ctl(n, o, m) ... | cs | p' \rangle$$

Execution of `pop` expects a message of the form $(n_1 n_2 ..., d)$ on the stack, from the header of which the leading name is removed, returning a pair consisting of this name and the remainder of the message. Recall that the format of names is specific to the NPO, and so $n_1$ can

be a suitable prefix of the address stack, not merely the topmost address. Executing `push` expects a message and a name $x$, which is prepended to the message header to yield the resulting message. The `lookup` operation expects a name $n$ on the stack, and the mapping associated with it in context state $cs$ is returned. The `update` primitive expects a name $n$ and a mapping $m$ on the stack, which it uses to change the context state. The notation $cs[n \mapsto m]$ describes the context state that is identical to $cs$ except that now name $n$ is associated with mapping $m$. The `create` primitive takes an explicit opcode argument $o$, and expects a name $n$ and mapping $m$ on the stack. The result is a constructed message $ctl(n,o,m)$ which is placed on the stack for further processing.

The opcodes with non-local effect are those for communication. These may be described by rules that describe transfer of a message at one NPO to another:

$$np_1[\langle msg...|cs_1|\texttt{send}(np_2);p_1'\rangle],\ np_2[\langle ...|cs_2|\texttt{receive};p_2'\rangle] \longrightarrow np_1[\langle ...|cs_1|p_1'\rangle], np_2[\langle msg...|cs_2|p_2'\rangle]$$

provided $np_1, np_2$ can communicate directly. The notation $np[\ldots]$ indicates the state at NPO $np$, and in the rule the two communicating NPOs are juxtaposed. In this rule, we have presented a synchronous transfer of the message between the NPOs. However, for other semantics, we can interpose a suitable abstract medium with appropriate semantics (synchronous/asynchronous, queue/bag, lossy/ideal), which can be modelled either abstractly or explicitly.

### 3.3 Proof techniques

We separate notions of *partial correctness* or "safety" from those of termination or *progress*. The former properties describe that nothing wrong happens during protocol execution, e.g., that no message is incorrectly forwarded to an unintended NPO, or that no name is interpreted in an incorrect context. These are fairly challenging to establish in the presence of dynamic changes to the routing and forwarding state of the network [1], and require *coinductive techniques* to show that essential structural properties of the context state are maintained, i.e., are *invariant*. A typical invariant is that the context states for forwarding eventually form an acyclic directed graph. Other important properties that need to be shown are that updates due to messages maintain the necessary consistency of how names are interpreted in the context state. Another typical requirement is the existence of default forwarding actions which ensure deliverability.

We posit that the notion of scope will be useful in establishing invariants for proving the correctness of various protocols. For example, in IP mobility support, the scope corresponding to a home "IP address" may be considered as comprising those NPOs which will resolve this name to the correct current location of the device. While this set of NPOs can dynamically change due to mobility, the abstractly characterized scopes satisfy invariant properties such as: (i) they include the router at the 'home address"; (ii) that IP messages reaching a member of this scope remain confined to it; and (iii) that the forwarding tables will always take a message to a member of the abstractly characterized scope.

## 4 Conclusions

We believe that a carefully chosen conceptual framework of communication primitives not only provides a clear understanding of the current, complex Internet, but also serves as the basis for a formal model of its semantics. We have presented such a framework and outlined its operational semantics. In future work, we propose to further extend this formal analysis, and also implement a 'universal forwarding engine' based on our primitives.

## References


[1] R. M. Amadio and S. Prasad. Modelling IP Mobility. *Journal of Formal Methods in System Design*, 17(1):61–99, 2000.

[2] H. Balakrishnan et al. A Layered Naming Architecture for the Internet. In *Proceedings of SIGCOMM 2004*, pages 343–352.

[3] D. Clark. The Design Philosophy of the Darpa Internet Protocols. In *Proceedings of SIGCOMM 1988*, pages 106–114.

[4] T. G. Griffin and J. L. Sobrinho. Metarouting. In *Proceedings of SIGCOMM 2005*, pages 1–12.

[5] M. Gritter and D. Cheriton. An Architecture for Content Routing Support in the Internet. In *Proceedings of USITS 2001*, pages 37–48.

[6] C. A. R. Hoare An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, 1969.

[7] P. J. Landin. The Mechanical Evaluation of Expressions. *Computer Journal*, 6(4):308–320, 1964.

[8] B. Loo et al. Declarative Routing: Extensible Routing with Declarative Queries. In *Proceedings of SIGCOMM 2005*, pages 289–300.

[9] C. Partridge et al. RFC 1546 - Host Anycasting Service, November 1993.

[10] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[11] J. H. Saltzer. Naming and Binding of Objects. In Rudolph Bayer et al. (eds.), *Operating Systems - An Advanced Course*, pages 99–208. Springer LNCS 60, 1978.

[12] I. Stoica et al. Internet Indirection Infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, April 2004.