Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 312 (2015) 3–17

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# Network-Conscious $\pi$-calculus - A Model of Pastry [1]

Ugo Montanari[2,4] Matteo Sammartino[2,3]

**Abstract**

A peer-to-peer (p2p) system provides the networking substrate for the execution of distributed applications. It is made of peers that interact over an *overlay network*. Overlay networks are highly dynamic, as peers can join and leave at any time. Traditional process calculi, such as $\pi$-calculus, CCS and others, seem inadequate to capture these kinds of networks, their routing mechanisms, and to verify their properties. In order to model network architecture in a more explicit way, in [14,18,15] we have introduced the *Network Conscious $\pi$-calculus* (`NCPi`), an extension of the $\pi$-calculus with names representing network nodes and links. In [15] (a simpler version of) `NCPi` has been equipped with a coalgebraic operational models, along the lines of Fiore-Turi presheaf-based approach [6], and with an equivalent History Dependent Automaton [13], i.e., an (often) finite-state automaton suitable for verification. In this paper we first give a brief account of these results. Then, our contribution is the sketch of a `NCPi` representation of the p2p architecture `Pastry`. In particular, we give models of its overlay network and of a Distributed Hash Table built on top of it, and we give evidence of their correctness by proving convergence of routing mechanisms.

*Keywords:* Peer-to-peer, routing, overlay network, distributed hash-table, pastry, verification, routing convergence, process calculus, network conscious pi-calculus, presheaf, coalgebra, HD-automaton

## 1 Introduction

A peer-to-peer (p2p) system provides the networking substrate for the execution of distributed applications. It is made of *peers* that interact over an application-level *overlay network*, built on top of the physical one. An overlay network is highly dynamic, as peers can join and leave it at any time, and this causes continuous reconfigurations of its topology. A key property is that routing mechanisms should work even after every reconfiguration.

Traditional process calculi, such as $\pi$-calculus [12], CCS [11] and others, seem inadequate to describe these kinds of networks, their routing mechanisms, and to

[1] Research supported by the EU Integrated Project 257414 ASCENS and by the Italian MIUR Project CINA (PRIN 2010)

[2] University of Pisa
Department of Computer Science
Pisa, Italy

[3] Email: sammarti@di.unipi.it

[4] Email: ugo@di.unipi.it

http://dx.doi.org/10.1016/j.entcs.2015.04.002

verify their properties. In fact, they abstract away from network details, as two processes are allowed to communicate only through shared channels.

In order to model network architecture in a more explicit way, in [14,18,15] we have introduced the *Network Conscious π-calculus* (NCPi), a seamless extension of the π-calculus with a natural notion of network: nodes and links are regarded as computational resources that can be created, passed and used to transmit, so they are represented as names, following the π-calculus methodology. The main features of NCPi are the following.

- There are two types of names: *sites*, which are the nodes of the network, and *links*, named connectors between pairs of sites. Sites are just atoms, e.g. $a$, links have the form $l_{ab}$, meaning that there is a link named $l$ from $a$ to $b$.
- The syntax can express the *creation* of a link through the restriction operator, and the *activation* of a transportation service over a link through a dedicated prefix. Separating these operations agrees with the π-calculus, where creating and using a channel as subject are two distinct operations. Moreover, processes are not required to communicate on shared channels: an extended output primitive is introduced that specifies emission and *destination* sites.
- Observations of the semantics represent concurrent transmissions in the form of multisets of routing paths. This follows the intuition that processes should act in a truly distributed manner, without a central coordinator that imposes an interleaving order to their actions. The associated behavioral equivalence is closed with respect to all operators of the language, including input prefix, i.e. it is a *congruence*. Moreover, the operational semantics is equipped with a mechanism for controlling the inference of routing paths according to a user-defined strategy. This allows for the implementation of routing algorithms.

Conveniently, in [15] we have introduced a presheaf-based coalgebraic semantics for NCPi, in the style of [6]. The basic idea of [6] is having a model where we distinguish: (a) a domain of resources, (b) a domain of programs and a (c) domain of "maps" between resources and programs. In NCPi, resources of a process are its free sites and links, describing its communication network. Therefore, (a) is a category $\mathbf{G}$ of suitable graphs, representing networks, equipped with endofunctors that add new vertices and edges, modeling network resources allocation; (b) is **Set**, where some objects are regarded as sets of NCPi processes; (c) is the category of functors $\mathbf{G} \to \mathbf{Set}$ (*presheaves on* $\mathbf{G}$), associating to each network the set of NCPi processes with such network. The operational semantics, then, is modeled as a coalgebra [17] with states in a presheaf, thus each state is decorated with its networks: this enables the explicit representation of network resource allocation along transitions. Unfortunately, we still have infinitely many states, because allocated resources may grow indefinitely, even if only a finite portion of them is actually accessible, e.g., in recursive processes performing extrusions. However, our presheaf of states is "well-behaved", so, according to [1], it is always possible to deallocate the unused resources and an equivalent *History Dependent (HD) automaton* [13] can be derived from the NCPi coalgebra. HD-automata are automata with allocation and deallocation along

transitions. They admit minimal, possibly finite state, representatives, where all bisimilar states are identified, which can be computed as shown and implemented in [5].

Section 2 of this paper is devoted to a recapitulation of NCPi syntax and semantics. The main contribution is Section 3 where, in order to demonstrate the expressive power of our language, we present a NCPi model of the p2p architecture Pastry [16]. This model has been devised in the context of the FP7 *Autonomic Service-Component Ensembles* (ASCENS) project, where the routing functionalities of Pastry are employed in a cloud-computing case study. Details can be found in [18, Chapter 6].

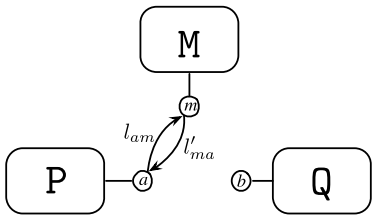
In Pastry, peers have unique identifiers, logically ordered in a ring. The main operation is routing by key: given a message and a target key, the message is delivered to the peer whose identifier is numerically closest to the key. Pastry is typically used for implementing *Distributed Hash Tables* (DHT), that are hash tables whose entries are distributed among peers: routing by key in this context amounts to hash table lookup. Our Pastry model is as follows. We begin by formalizing the features of Pastry routing that ensure its convergence. These are informally stated in [16], but we need a rigorous formulation so that we can prove the correctness of our model. Then we give a NCPi implementation of a Pastry peer. The basic idea is capturing identifiers as sites, and the overlay network as a collection of links between peers. We model peers' routing data structures and their operations, reconfiguration due to joining peers, and the provision of routing functionalities to applications. Node joins trigger a complex procedure, ending up with the creation of new links from/to the joining peer. We show that the resulting overlay still guarantees routing convergence. Finally, we model a simple DHT, where lookups are represented as routing paths from the peer that invoked the lookup to the one responsible for the target key. These paths are derived by composing atomic forwarding services offered by peers, employing the mechanism provided by the model to implement routing protocols. We prove that we have routing convergence also in this scenario, i.e., lookups always reach the correct peer.

## 2 Network Conscious $\pi$-calculus

### *2.1 Illustrative example*

In order to have a closer look at the calculus, consider the system in Fig. 1. Its aim is modeling a network whose topology is determined at run-time. We have a network manager M, capable of creating new links and granting access to them, and two processes P and Q, which access the network through $a$ and $b$ respectively; they are willing to communicate, but no links exists between $a$ and $b$, so P will ask M to create such link. Finally, we have a link server L which which is able to offer a transportation service over a given link.

The actual definition says that M can receive two sites at $m$, create a new link between them and send it from $m$ to the first of the received sites. Notice that the output prefix has three components, from left to right: emission site, destination




$$\begin{aligned}
\mathtt{M} &\stackrel{\text{def}}{=} m(x).m(y).(l_{xy})(\overline{m}xl_{xy}.\mathtt{M}) \\
\mathtt{P} &\stackrel{\text{def}}{=} \overline{a}ma.\overline{a}mb.a(l_{(xy)}).(\overline{a}bc.\mathtt{P}' \mid \mathtt{L}(l_{xy})) \\
\mathtt{Q} &\stackrel{\text{def}}{=} b(x).\mathtt{Q}' \\
\mathtt{L}(l_{xy}) &\stackrel{\text{def}}{=} l_{xy}.\mathtt{L}(l_{xy}) \\
\mathtt{S} &\stackrel{\text{def}}{=} \mathtt{P} \mid \mathtt{M} \mid \mathtt{Q} \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma})
\end{aligned}$$

Fig. 1. Example system.

site and datum.

The process P can first send $a$ and $b$ from $a$ to $m$ and then wait for a link at $a$. This link is received together with its endpoints, as $a$ and $b$ are bound in $(l_{(ab)})$. The process then becomes the parallel composition of two components: the first one can send $c$ from $a$ to $b$; the second one invokes the process L, whose function is activating the link $l_{ab}$. This activation is expressed as the *link prefix* $l_{ab}.\_$ in the definition of L: when consumed, it spawns a *transportation service* over $l_{ab}$, which can be used by the execution context (i.e.,, by other processes in parallel) to forward a datum from $a$ to $b$. The link prefix expresses a single activation of the link, as input/output prefixes in the $\pi$-calculus express a single usage of their subject channel. This explains the recursive definition of L, which is intended to model a persistent connection.

The process Q simply waits for a datum at $b$. Finally, the whole system S is the parallel composition of P, Q, M and two processes modeling a bidirectional persistent connection between $a$ and $m$.

Before showing some steps of computation, we briefly introduce observations by comparison with the $\pi$-calculus. As in the $\pi$-calculus, we have observations representing inputs, output and complete communications. However, since NCPi allows for remote communications, they all include the (possibly empty) sequence of links that are traversed in the communication. For instance, the process P can emit $a$ at $a$, with destination $m$, as follows

$$\mathtt{P} \xrightarrow{\bullet;\overline{a}ma} \overline{a}mb.a(l_{(xy)}).(\overline{a}bc.\mathtt{P}' \mid \mathtt{L}(l_{xy}))$$

The label $\bullet;\overline{a}ma$ is a zero-length (i.e., with empty sequence of links) *output path*. The symbol $\bullet$ is syntactic sugar, indicating where the path starts. In general, there may be a list of links $W$ between $\bullet$ and $\overline{a}ma$, meaning that $a$ went through $W$ before being emitted.

Symmetrically, M can receive the datum $a$, with destination $m$, at $m$

$$\mathtt{M} \xrightarrow{mma;\bullet} m(y).(l_{ay})\overline{m}al_{ay}.\mathtt{M}$$

where $mma;\bullet$ is a (zero-length) *input path*. The first two sites in $mma$, namely

reception and destination site respectively, coincide because the path represents a local input, analogous to the early $\pi$-calculus input action $ma$. Here the presence of a list $W$ of links between $m'ma$, for any $m'$, and $\bullet$ would mean that $W$ can be employed to reach the destination $m$ from $m'$.

Before introducing paths denoting complete communications, we introduce *service paths*, which have no counterpart in the $\pi$-calculus. A service path has the form $a; W; b$, where $W$ is a sequence of links. It represents a transportation service that can be used by the execution context to route a datum from $a$ to $b$. For instance we have

$$\mathtt{L}(l_{am}) \xrightarrow{a;l_{am};m} \mathtt{L}(l_{am})$$

where $a; l_{am}; m$ is a transportation service from $a$ to $m$ over $l_{am}$.

Finally, we have complete communication, denoted by a *complete path* $\bullet; W; \bullet$. Unlike the $\pi$-calculus $\tau$-action, this observation is not silent, as we allow the path $W$ of the transmitted datum to be observed; the datum itself remains unobservable. Another difference is that a complete path is usually produced by more than one synchronization, each one concatenating a compatible pair of paths. For instance, in order for $\mathtt{P}$ to communicate $a$ to $\mathtt{M}$, there must be a first synchronization between $\mathtt{P}$ and $\mathtt{L}(l_{am})$

$$\mathtt{P} \mid \mathtt{L}(l_{am}) \xrightarrow{\bullet;l_{am};\overline{m}ma} \dots$$

where $\bullet; l_{am}; \overline{m}ma$ is the concatenation of $\bullet; \overline{a}ma$ and $a; l_{am}; m$. These paths are compatible because the former emits its datum at the site from which the latter forwards data, namely $a$. Their concatenation represents the forwarding of $a$ from $a$ to $m$ using $l_{am}$ (the new emission site, in fact, is $m$). Here the continuation is the parallel composition of the continuations shown above. A complete path is produced by another, final synchronization between $\mathtt{P} \mid \mathtt{L}(l_{am})$ and $\mathtt{M}$

$$\mathtt{P} \mid \mathtt{L}(l_{am}) \mid \mathtt{M} \xrightarrow{\bullet;l_{am};\bullet} \dots$$

meaning that a complete communication over $l_{am}$ has happened.

Now we overview the steps the entire system $\mathtt{S}$ can perform:

(i) $\mathtt{P}$ **communicates to** $\mathtt{M}$ **the endpoints** $a$ **and** $b$ **of the link to be created:** it is observed as two consecutive occurrences of $\bullet; l_{am}; \bullet$. The state of the system after this interaction is

$$a(l_{(xy)}).(\overline{a}bc.\mathtt{P}' \mid \mathtt{L}(l_{xy})) \mid (l_{ab})(\overline{m}al_{ab}.\mathtt{M}) \mid \mathtt{Q} \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma}) \ .$$

(ii) $\overline{m}al_{ab}.\mathtt{M}$ **communicates** $l_{ab}$ **to** $a(l_{(xy)}).(\overline{a}bc.\mathtt{P}' \mid \mathtt{L}(l_{xy}))$**:** the observation is $\bullet; l'_{ma}; \bullet$, and the resulting state is

$$(l_{ab})(\overline{a}bc.\mathtt{P}' \mid \mathtt{L}(l_{ab}) \mid \mathtt{M}) \mid \mathtt{Q} \mid \mathtt{L}(l_{am}) \mid \mathtt{L}(l'_{ma})$$

where the scope of $l_{ab}$ has been extended.

(iii) $\overline{a}bc.\mathtt{P}'$ **communicates** $c$ **to** $\mathtt{Q}$**:** in this case, despite $l_{ab}$ is used for the transmission, only $\bullet; \bullet$ can be observed, because such link is restricted. This is analogous

to the $\pi$-calculus $\tau$ action. The continuation is

$$(l_{ab})(\mathtt{P}' \,|\, \mathtt{L}(l_{ab}) \,|\, \mathtt{M}) \,|\, \mathtt{Q}'[c/x] \,|\, \mathtt{L}(l_{am}) \,|\, \mathtt{L}(l'_{ma}) \quad .$$

### *2.2 Overview of the calculus*

Here we give a brief overview of the calculus. We assume to have two enumerable sets $\mathcal{S}$ and $\mathcal{L}$ of sites and links, respectively; $\mathcal{L}$ is equipped with two functions $s, t \colon \mathcal{L} \to \mathcal{S}$, telling source and target of each link. We denote by $l_{ab}$ a link $l$ such that $s(l) = a$ and $t(l) = b$.

**Definition 2.1** `NCPi` processes are defined as follows, for $a, b \in \mathcal{S}, l_{ab} \in \mathcal{L}$:

$$\begin{array}{llllllllll} p ::= \mathbf{0} & | & \pi.p & | & p+p & | & p \,|\, p & | & (r)p & | & A(r_1, r_2, \ldots, r_n) \\ r ::= a & | & l_{ab} & & s ::= a & | & l_{(ab)} & & \pi ::= \overline{a}br & | & a(s) \quad | \quad l_{ab} \quad | \quad \tau \end{array}$$

$$A(s_1, s_2, \ldots, s_n) \stackrel{\text{def}}{=} p \qquad i \neq j \implies n(s_i) \cap n(s_j) = \varnothing$$

where $n(a) = \{a\}$ and $n(l_{ab}) = n(l_{(ab)}) = \{l_{ab}, a, b\}$.

The syntax of `NCPi` processes extends the $\pi$-calculus one as follows. A restriction $(r)p$ can bind either a site ($r \in \mathcal{S}$) or a link ($r \in \mathcal{L}$). The output prefix $\overline{a}br$, besides the emission site $a$ and the datum $r$, also specifies a destination site $b$. Inputs $a(l_{(ab)})$ can express the reception of a link *and* its endpoints. The *link prefix* $l_{ab}.p$ means that this process can offer to the execution context the service of transporting a datum from $a$ to $b$ through $l$ and then can continue as $p$. Formal parameters of process definitions must not share names.

The free names $\mathrm{fn}(p)$ are defined by recursion as expected. We briefly describe some interesting cases. If $l_{cd}$ is not argument of a top level restriction or input binder in $p$, namely $p$ is of the form $\overline{a}bl_{cd}.p'$ or $l_{cd}.p'$, then $\mathrm{fn}(p)$ includes $\{l_{cd}, c, d\}$; otherwise, $l_{cd}$ is bound but its endpoints may be free, for instance $\mathrm{fn}((l_{cd})p') = \{c, d\} \cup \mathrm{fn}(p') \setminus \{l_{cd}\}$. The case of bound sites, for instance $p = (a)p'$, requires some care, because free links in $p'$ with endpoint $a$ become implicitly bound in $p$; to avoid this, we rule out some non-well-formed processes (see [15, 3.1] for details).

Structural congruence for processes include usual commutative monoidal laws for $|$ and $+$, $\alpha$-conversion w.r.t. both bound sites and links, and unfolding of process definitions.

In the previous section we have seen single routing paths as observations, but the semantics allows observing several paths in parallel, in the form of *multisets* of paths. For instance, we can observe `S` in Fig. 1 doing $\bullet; \overline{a}ma \,|\, a; l_{am}; m \,|\, mma; \bullet$, which represents a three-element multiset.

**Definition 2.2** Paths (denoted $\alpha$) and multisets of paths (denoted $\Lambda$) are defined

as follows:

$$
\begin{array}{lll}
\alpha ::= a; W; b & & \text{(Service path)} \\
\quad \mid \ \bullet; W; \bullet & & \text{(Complete path)} \\
\quad \mid \ \bullet; W; \overline{a}br & & \text{(Output path)} \\
\quad \mid \ abr; W; \bullet & & \text{(Free input path)} \\
\quad \mid \ ab(s); W; \bullet & n(s) \cap (n(W) \cup \{a, b\}) = \varnothing & \text{(Bound input path)}
\end{array}
$$

$$
r ::= a \mid l_{ab} \qquad s ::= a \mid l_{(ab)} \qquad W ::= l_{ab} \mid W; W \mid \epsilon
$$

$$
\Lambda ::= 1 \mid \alpha \mid \Lambda_1 | \Lambda_2 \mid (r)\Lambda
$$

Besides paths presented in the previous section, we have a bound input path, representing the reception of a bound name. Multisets of paths can be: the *empty* multiset 1; the *singleton* $\alpha$; the *union* $\Lambda_1 \mid \Lambda_2$ and the *extrusion* $(r)\Lambda$. Notice that we do not have a bound output path; instead, an extrusion restriction $(r)$ can have a whole multiset $\Lambda$ in its scope, because we allow many output paths in $\Lambda$ to extrude $r$.

We have some structural congruence axioms for paths and multisets: paths $\alpha$ are strings, i.e., they form a monoid with multiplication ; and unit $\epsilon$; $\Lambda$ are multisets, i.e., they form a commutative monoid with multiplication $\mid$ and unit 1; extrusion restrictions can be swapped and their scope can be extended to include multisets where restricted names do not occur free.

The `NCPi` LTS is the smallest one derived from a collection of SOS rules. We describe the most interesting rules, namely those implementing process synchronization. Synchronization is performed in two steps:

**Paths collection:** paths of parallel processes are collected through the following rule

$$
\frac{p_1 \xrightarrow{\Lambda_1} q_1 \quad p_2 \xrightarrow{\Lambda_2} q_2}{p_1 \mid p_2 \xrightarrow{\Lambda_1 \mid \Lambda_2} q_1 \mid q_2}
$$

where bound names in $\Lambda_1$ are required to be fresh w.r.t. $p_2$ and $\Lambda_2$ (the same for $\Lambda_2$, $p_1$ and $\Lambda_1$);

**Paths concatenation:** other rules pick two compatible paths from the multiset produced by the previous step and replace them with their concatenation, without modifying the source process; in other words, these rules synchronize two subprocesses of the source process. For instance, an output path and a service path with a common endpoint can be concatenated using the following rule, resulting in an extended output path

$$
\frac{p \xrightarrow{(R)\ (\bullet; W; \overline{a}br \mid a; W'; c \mid \Theta)} q}{p \xrightarrow{(R)\ (\bullet; W; W'; \overline{c}br \mid \Theta)} q}
$$

where $(R)$ is a sequence of restrictions and $\Theta$ is a concurrent path without extrusion restrictions (they have all been brought at the top level using scope extension

of path multisets).

The bisimilarity for the NCPi LTS is defined as follows. Its simple definition resembles the $\pi$-calculus early bisimulation because, as mentioned in section 2.1, we adopt early style inputs.

**Definition 2.3** *A binary, symmetric and reflexive relation $R$ is a* network conscious bisimulation *if $(p,q) \in R$ and $p \xrightarrow{\Lambda} p'$, with* $\mathrm{bn}(\Lambda)$ *fresh w.r.t. $q$, implies that there is $q'$ such that $q \xrightarrow{\Lambda} q'$ and $(p',q') \in R$. The bisimilarity is the largest such relation and is denoted by $\sim^{NC}$.*

The relation $\sim^{NC}$ enjoys the following important property.

**Theorem 2.4** *$\sim^{NC}$ is a congruence with respect to all NCPi operators.*

Intuitively, this property holds because observations $\Lambda$ allow distinguishing processes with different level of parallelism (e.g., a parallel process and its sequential interleavings).

NCPi has a built-in mechanism to control the inference of paths. In fact, SOS rules derive all possible paths, non-deterministically. In order to select only specific paths, e.g. according to a specific routing strategy, one can define a *forwarding predicate*

$$\varphi\colon \mathcal{L} \times \mathcal{S} \times Proc \to \{\texttt{true}, \texttt{false}\}$$

and then use it as an additional side condition for rules that perform the path concatenation step: if $\varphi(l_{ab}, c, p)$ returns `true` then a path of $p$, with destination $c$, can be concatenated with $l_{ab}$. In this way, for instance, we could exclude non-optimal links according to some metric (cost, latency, distance, and others). See [15, 6] for a forwarding predicate modeling the *Border Gateway Protocol*.

# 3 Pastry model

In this section we use NCPi to model Pastry overlay networks and *Distributed Hash Tables* (DHTs).

## *3.1 Pastry overview*

Pastry is a peer-to-peer architecture where peers and keys have *identifiers*, regarded as arranged in clockwise order on a *ring*. The main service provided by Pastry is *routing by key*: given a key $k$, Pastry delivers the message to the peer which is *responsible for* $k$, i.e. the one whose identifier is numerically closest to $k$ than all other peers. Routing is implemented as follows. Each peer with identifier $id$ maintains two data structures: a *routing table* and a *leaf-set*. The routing table contains peers that share a prefix with $id$. The leaf-set contains peers (*leaves*) with numerically closest smaller and larger identifiers, relative to $id$. Whenever $id$ receives a message with target key $k$, it checks whether $k$ belongs to the leaf-set range. If so, the message is forwarded to the leaf numerically closest to $k$ (if such leaf is not $id$ itself). Otherwise, the routing table is used: the next hop is the peer sharing the longest prefix with $k$. An example system is shown in Fig. 2, where identifiers are binary strings.

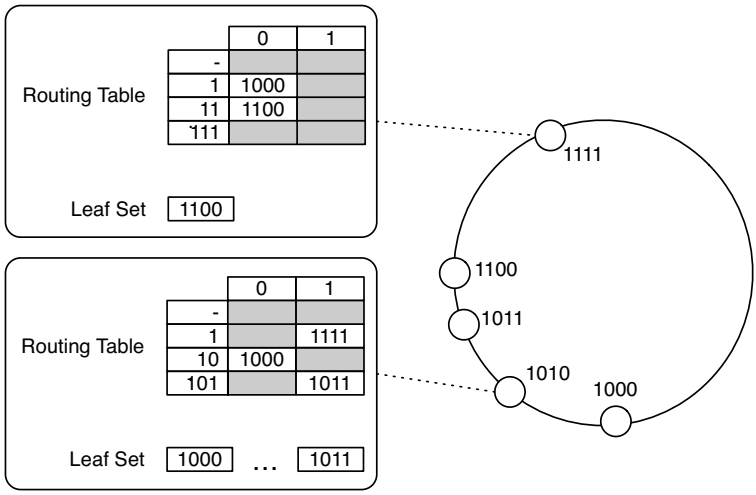



Fig. 2. `Pastry` example system.

**Example 3.1** *Consider the peer with identifier* 1010 *in Fig. 2, and suppose* 1100 *is responsible for the key* 1101*. A message from* 1010 *with target key* 1101 *is routed as follows. Since* 1101 *does not belong to the interval* $[1000, 1011]$ *spanned by the leaf-set of* 1010*, the routing table is used: the longest prefix shared by* 1010 *and* 1101 *is* 1*, so the message is forwarded to the peer in the cell* $(1, 1)$*, namely* 1111*. Once* 1111 *receives the message, it discovers that* 1101 *is in its leaf-set range (the leaf-set has* 1111 *itself as upper bound, as there are no peers with larger identifiers), so it forwards the message to the leaf closest to* 1101*, that is* 1100*.*

One important property of `Pastry` routing procedure is convergence: the message eventually gets to its destination. This property is stated [16, 2.2], but only in informal terms. Here we provide a formal, mathematical account of it, in order to prove the correctness of our model. Let $x, y$ two identifiers. We define the *ring distance* $d_r$ between $x$ and $y$ as the number of identifiers between $x$ and $y$ on the ring. Formally, if $I$ is the size of the space of identifiers, we have

$$d_r(x, y) = \begin{cases} I - |x - y| & |x - y| > \lfloor I/2 \rfloor \\ |x - y| & \text{otherwise} \end{cases}$$

The first case happens when $x$ and $y$ have numerical distance greater than half the ring: then we must consider the complementary arc.

**Property 3.2 (Routing convergence)** *The routing procedure always converges: given a message with target key* $k$ *and a peer id, either id is responsible for* $k$ *or it can forward the message to* $id'$ *such that* $d_r(id', k) < d_r(id, k)$*.*

### *3.2 Peer model*

The key idea is modeling identifiers as sites, and the routing table and the leaf-set of a peer as two collections of links $\mathcal{L}_{RT}$ and $\mathcal{L}_{LS}$, which form the overlay network of a peer. Notice that these links are *logical*, not physical connections: the

overlay network is an abstraction of the underlying physical one. When proving convergence of routing, we assume that failures happening at the physical level are handled locally.

We denote by $a \boxdot b$ a link to $b$ in $a$'s routing table and by $a \boxminus b$ a link to $b$ in $a$'s leaf-set. A peer with identifier $a$ is modeled as the process

$$\begin{aligned} \texttt{Peer}(a, \mathcal{L}_{RT}, \mathcal{L}_{LS}) &\stackrel{\text{def}}{=} (\mathcal{O}_{RT})(\mathcal{O}_{LS})\, \texttt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) \\ &\quad \mid \texttt{RT}(\mathcal{L}_{RT}, \mathcal{O}_{RT}) \mid \texttt{LS}(\mathcal{L}_{LS}, \mathcal{O}_{LS}) \\ \texttt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) &\stackrel{\text{def}}{=} \texttt{JoinH}(a) + \texttt{Route}(\mathcal{O}_{RT}, \mathcal{O}_{LS}) \end{aligned}$$

Processes `RT` and `LS` allow querying and modifying routing table and leaf-set. These operations are called internally via the names in $\mathcal{O}_{RT}$ and $\mathcal{O}_{LS}$. The process `Control` implements the control logic of a peer.

`JoinH` executes the distributed protocol for node joins as follows. When a peer $b$ is willing to join the network, it sends a join message including its identifier trough a "bootstrap" peer. This message is routed through the overlay as a message with target key $b$, i.e., the target peer is the one with identifier closest to $b$. More precisely, at each hop the peer $c$ receiving the join message checks whether $b$ is in its leaf-set range: if so, $c$ forwards the join message to the leaf $l$ closest to $b$ using $c \boxminus l$; otherwise, it looks for the peer $r$ in its routing table that has the longest common prefix with $b$, and uses $c \boxdot r$ to forward the join message to $r$. Finally, $c$ sends the content of its routing table to $b$. The last peer to receive the join message, namely the one closest to $b$, also sends its leaf-set, because some of its leaves will become leaves of $b$. Upon receiving these messages, $b$ creates new links towards the peers referred by the messages. These links form the routing table and leaf-set of $b$. Finally, $b$ notifies its existence to the peers in its data structures; these peers update their routing information accordingly. Now $b$ is part of the overlay network. In [18, Theorem 6.3.1] we give a detailed implementation of the join procedure and we prove that reconfiguration of the overlay network due to node joins preserves Property 3.2.

The process `Route` makes the overlay network of $a$ available to applications. To do this, we introduce a special service path $a; a \rhd b \uparrow; b$ ($\rhd \in \{\boxminus, \boxdot\}$), which is observed when `Route` consumes a link prefix, where the link belongs to $a$'s routing table or leaf-set. The symbol $\uparrow$ means that the link is intended to be used by the applications running on top of `Pastry`.

A `Pastry` system is modeled as the parallel composition of peer processes. For the system in Fig. 2 we have

$$\texttt{Sys} \stackrel{\text{def}}{=} \texttt{Peer}(1000) \mid \texttt{Peer}(1010) \mid \texttt{Peer}(1011) \mid \texttt{Peer}(1100) \mid \texttt{Peer}(1111)$$

### *3.3 DHT model*

Now we want to model routing behavior for a simple Distributed Hash Table (DHT), where observations are routing paths of DHT lookups. In order to do this, we

$$\varphi_{\texttt{Pastry}}(l_{ab}, k, p) = \textbf{case } l_{ab} \textbf{ of}$$

$$\textbf{if } a \boxminus b \Rightarrow \left( \begin{array}{c} \forall b' \neq b : p \xrightarrow{a;a \boxminus b' \uparrow; b'} p' \implies d_r(k,b) < d_r(k,b') \\ \bigwedge \\ \exists b_1, b_2 : \left( \begin{array}{c} p \xrightarrow{a;a \boxminus b_1 \uparrow; b_1} p_1, p \xrightarrow{a;a \boxminus b_2 \uparrow; b_2} p_2 \\ \wedge \\ b_1 \preceq a \preceq b_2 \ \wedge \ b_1 \prec k \prec b_2 \end{array} \right) \end{array} \right)$$

$$\textbf{if } a \boxdot b \Rightarrow \left( \begin{array}{c} \forall b' \neq b : p \xrightarrow{a;a \boxminus b' \uparrow; b'} p' \implies d_r(k,b) < d_r(k,b') \\ \wedge \\ shl(b,k) > shl(a,k) \end{array} \right)$$

$$\textbf{if } a \trianglerighteq b \Rightarrow b = k$$

Fig. 3. `Pastry` forwarding predicate.

introduce a new type of link: $a \trianglerighteq k$ means that the peer with identifier $a$ is responsible for the key $k$.

The `Pastry` routing strategy is implemented through the forwarding predicate $\varphi_{\texttt{Pastry}}$, shown in Fig. 3. We denote by $shl(x, y)$ the length of the longest prefix shared by $x$ and $y$, and by $\prec$ the order relation on identifiers, seen as natural numbers. The first case allows forwarding a message with target $k$ from $a$ to $b$, via a link in $a$'s leaf-set, provided that: (i) there is no other leaf $b'$ which is closer to $k$ than $b$; (ii) $a$ has two leaves $b_1$ and $b_2$, on opposite sides of (but not necessarily distinct from) $a$, and (iii) $k$ is between them, i.e. $k$ is within the leaf-set range. The second case allows a forwarding through a link in the routing table whenever there is no better link in the leaf-set and the identifier $b$ of the reached peer shares (at least) one more digit with $k$ than $a$. The third case treats links that allow reaching a key $k$ via the peer responsible for it: it is required that the the link's target is indeed $k$. Notice that the routing mechanism is the same as the join procedure. There observations are single hops, because some operations need to be performed at each forwarding. Here, instead, a single observation describes all the routing steps.

We can model a Distributed Hash Table over a `Pastry` system made of peers $a_1, \ldots, a_n$ as follows. Suppose the DHT has $m$ key-value pairs $\langle k_i, v_i \rangle$, and let $a_{k_i}$ be

the identifier of the peer responsible for $k_i$, i.e. the closest to $k_i$ among $a_1, \ldots, a_n$.

$$\begin{aligned} \texttt{DHT} &\stackrel{\text{def}}{=} \texttt{Peer}(a_1) \mid \ldots \mid \texttt{Peer}(a_n) \mid \texttt{H} \\ \texttt{H} &\stackrel{\text{def}}{=} \texttt{Entry}(k_1, v_1, a_{k_1}) \mid \ldots \mid \texttt{Entry}(k_m, v_m, a_{k_m}) \\ \texttt{Entry}(k, v, a) &\stackrel{\text{def}}{=} a \trianglerighteq k \mid k(b).\overline{a}bv.\texttt{Entry}(k, v, a) \end{aligned}$$

Here H represents the DHT content as the parallel composition of processes that handle the table's entries. The idea is implementing a DHT lookup request for a key $k$ as a message with destination $k$, carrying the identifier $b$ of the sender. Upon receiving this message, the handler $\texttt{Entry}(k, v, a)$ for $\langle k, v\rangle$ replies to $b$ with a message containing $v$. Notice that node joins in Pastry may introduce new key-value pairs in the DHT. However, for simplicity, we assume that the DHT is fixed. The addition of a key $k$ with value $v$ could be modeled by using the routing mechanism to find the peer with id $a_k$ closest to $k$, and then spawning a new process $\texttt{Entry}(k, v, a_k)$.

We provide an account of Property 3.2 in this scenario.

**Lemma 3.3** *For every peer $a$ and key $k$ there is $\texttt{DHT} \xrightarrow{a;a \rhd b;b} \texttt{DHT}'$, where $\rhd \in \{\boxminus, \boxdot\}$, such that either $b = k$ or $b$ is closer to $k$ than $a$, i.e. $a, b, k$ satisfy Property 3.2.*

The following result is an immediate consequence of Lemma 3.3 and of the definition of $\varphi_{\texttt{Pastry}}$. It says that, given a key $k$ and a peer $a$, there always is a path from $a$ routing a lookup request for $k$.

**Theorem 3.4** *Let $k$ be a key in the DHT and $a_k$ the peer responsible for it. Then, for every peer $a$, there exists a transition*

$$\texttt{DHT} \xrightarrow{a;a \rhd a_1;\ldots;a_n \rhd a_k;a_k \trianglerighteq k;\bullet} \texttt{DHT}'$$

*with $\rhd \in \{\boxminus, \boxdot\}$ and $n \geq 0$.*

As an example, we show how to compute a routing path in the system of Fig. 2. For simplicity, let us consider a DHT with only one key-value pair $(1101, v)$ located at 1100:

$$\texttt{H} \stackrel{\text{def}}{=} 1100 \trianglerighteq 1101 \mid 1101(a).\overline{1100}\, a\, v.\texttt{H} \qquad \texttt{DHT} \stackrel{\text{def}}{=} \texttt{Sys} \mid \texttt{H}$$

where Sys is defined in section 3.2. Consider the following process, representing a user application running at 1010

$$\texttt{App} \stackrel{\text{def}}{=} \overline{1010}\ 1101\ 1010.1010(v').\texttt{App}'(v') \ .$$

This process sends a lookup request for the key 1101, receives the result and uses it for some computations. So we have

$$\texttt{App} \xrightarrow{\bullet;\overline{1010}\ 1101\ 1010} 1010(v').\texttt{App}'(v') \ .$$

The routing steps for this request are those of Example 3.1. In this context, they

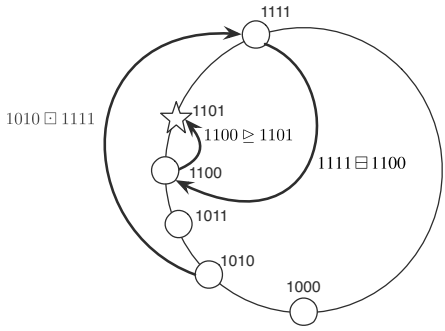



Fig. 4. Routing path from 1010 for the key 1101 in the system of Fig. 2.

become the following ones, depicted in Fig. 4

$$\texttt{Peer}(1010) \xrightarrow{1010;1010\boxdot 1111;1111} \texttt{Peer}(1010)$$
$$\texttt{Peer}(1111) \xrightarrow{1111;1111\boxminus 1100;1100} \texttt{Peer}(1111)$$

The predicate $\varphi_{\texttt{Pastry}}$, applied to $\texttt{Peer}(1010)$ and $1010 \boxdot 1111$, and to $\texttt{Peer}(1111)$ and $1111 \boxminus 1100$, for the key 1101, holds true, so we can use the SOS rules to concatenate the three paths shown so far (see the concatenation step described in section 2.2). The result is

$$\bullet; 1010 \boxdot 1111; 1111 \boxminus 1100; \overline{1100}\ 1101\ 1010$$

The complementary input path can be inferred

$$\texttt{H} \xrightarrow{1100\ 1101\ 1010;1100\trianglerighteq 1101;\bullet} \overline{1100}\ 1010\ v.\texttt{H}\ .$$

Finally, we can concatenate all these paths and get

$$\texttt{App} \mid \texttt{DHT} \xrightarrow{\bullet;1010\boxdot 1111;1111\boxminus 1100;1100\trianglerighteq 1101;\bullet} 1010(v').\texttt{App}(v') \mid \texttt{Sys} \mid \overline{1100}\,1010\,v.\texttt{H}$$

which exhibits the whole routing path from 1010 to 1100. Finally, assuming that the overlay network has a path back to 1010, the following configuration is reached

$$\texttt{App}(v) \mid \texttt{DHT}\ .$$

# 4 Conclusions

In this paper we presented NCPi, an extension of $\pi$-calculus with an explicit notion of network. To achieve this, we enriched the syntax with named connectors and defined a LTS semantics whose observations are multisets of routing paths. The concurrent nature of the semantics makes bisimilarity a congruence. We used NCPi to model the peer-to-peer architecture Pastry. The advantage is that it is possible to observe routing paths for DHT lookup operations as whole routing paths through the overlay, resulting from multiple synchronizations among peers.

Future work includes using `NCPi` to model other networking scenarios, for instance social networks. We also plan to extend our `Pastry` scenario by modeling a real-life application, for instance file-sharing, that employs DHT operations. We could also add quantitative information to links, e.g., costs, bandwidth, etc., in order to capture QoS or similar requirements.

The works most closely related to ours are [7] and [3] where network-aware extensions of $D\pi$ [10] and KLAIM [2], called respectively $D\pi_F$ and TKLAIM, are presented. Their network representations are quite different from ours: in $D\pi_F$ locations are explicitely associated with their connectivity via a type system, TKLAIM has a special process to represent connections, while in our calculus connections are just names, so the available network nodes and connections correspond to the standard notion of free names. This brings simpler primitives, but also a higher level of dinamicity: connections can be created and passed among processes, as shown in the illustrative example. As for our `Pastry` model, it would not be easily implementable in $D\pi_F$ and TKLAIM: network is always available in these calculi, whereas we control the activation of leaf-set and routing table links via the link prefix; $D\pi_F$ and TKLAIM do not allow observing multi-hop paths, whereas we are able to represent DHT lookups as routing paths through the overlay. We can also cite [8,9,4] as examples of calculi where resources carry some extra information. See [18, 7.1.1] for a detailed comparison.

## References

[1] Vincenzo Ciancia, Alexander Kurz, and Ugo Montanari. Families of symmetries as efficient models of resource binding. *ENTCS*, 264(2):63–81, 2010.

[2] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998.

[3] Rocco De Nicola, Daniele Gorla, and Rosario Pugliese. Basic observables for a calculus for global computing. *Inf. Comput.*, 205(10):1491 – 1525, 2007.

[4] Edsko De Vries, Adrian Francalanza, and Matthew Hennessy. Reasoning about explicit resource management (extended abstract). In *PLACES 2011*, pages 15–21. ETAPS, April 2011.

[5] Gian Luigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. *Theor. Comput. Sci.*, 331(2-3):325–365, 2005.

[6] Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *LICS 2001*, pages 93–104. IEEE Computer Society, 2001.

[7] Adrian Francalanza and Matthew Hennessy. A theory of system behaviour in the presence of node and link failure. *Inf. Comput.*, 206(6):711 – 759, 2008.

[8] Matthew Hennessy. A calculus for costed computations. *LMCS*, 7(1), 2011.

[9] Matthew Hennessy and Manish Gaur. Counting the cost in the picalculus (extended abstract). *ENTCS*, 229(3):117–129, 2009.

[10] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

[11] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[12] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I/II. *Inf. Comput.*, 100(1):1–77, 1992.

[13] Ugo Montanari and Marco Pistore. Structured coalgebras and minimal hd-automata for the $\pi$-calculus. *Theor. Comput. Sci.*, 340(3):539–576, 2005.

[14] Ugo Montanari and Matteo Sammartino. Network conscious $\pi$-calculus: A concurrent semantics. *ENTCS*, 286:291–306, 2012.

[15] Ugo Montanari and Matteo Sammartino. A network-conscious $\pi$-calculus and its coalgebraic semantics. *Theor. Comput. Sci.*, 546(0):188 – 224, 2014.

[16] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[17] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.

[18] Matteo Sammartino. *A Network-Aware Process Calculus for Global Computing and its Categorical Framework*. PhD thesis, University of Pisa, 2013. available at http://www.di.unipi.it/~sammarti/publications/thesis.pdf.