Università di Pisa
Dipartimento di Informatica

Technical Report: TR-12-01

Network Conscious π-calculus

Ugo Montanari Matteo Sammartino

February 3, 2012

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Network Conscious π-calculus

Ugo Montanari and Matteo Sammartino

Dipartimento di Informatica, Università di Pisa
{ugo,sammarti}@di.unipi.it

Abstract. Traditional process calculi usually abstract away from network details, modeling only communication over shared channels. They, however, seem inadequate to describe new network architectures, such as Software Defined Networks [1], where programs are allowed to manipulate the infrastructure. In this paper we present a network conscious, proper extension of the π-calculus: we add connector names and the primitives to handle them, and we provide both an interleaving and a concurrent semantics. The extension to connector names is natural and seamless, since they are handled in full analogy with ordinary names. In the interleaving case, observations are the routing paths through which sent and received data are transported, while in the concurrent case we allow to observe multisets of paths. However, restricted connector names do not appear in the observations, which thus can possibly be as abstract as in the π-calculus. Finally, for the concurrent semantics we show that bisimilarity is a congruence, and this property holds also for the concurrent version of the π-calculus.

1 Introduction

The trend in networking is going towards more "open" architectures, where the infrastructure can be manipulated in software. This trend started in the nineties, when OpenSig [2] and *Active Networks* [3] were presented, but neither gained wide acceptance due to security and performance problems. More recently, OpenFlow [4, 1] or, more broadly, *Software Defined Networking* has become the leading approach, supported by Google, Facebook, Microsoft and others. Software defined networks (SDNs) are networks in which a programmable *controller machine* manages a group of switches, by instructing them to install or uninstall forwarding rules and report traffic statistics.

Traditional process calculi, such as π-calculus [5, 6], CCS [7] and others, seem inadequate to describe these kinds of networks, because they abstract away from network details. In fact, two processes are allowed to communicate only through shared channels and it is not possible to express explicitely the fact that there is some complex connector between them. To give better visibility to the network architecture, in recent years network-aware extensions of known calculi have been devised, most importantly: $D\pi_F$ [8], based on the distributed π-calculus [9], and tKlaim [10], based on Klaim [11].

This paper focuses on the π-calculus, and aims at equipping it with a natural notion of network: nodes and connectors are computational resources, so

it is reasonable to represent them as (structured) names. We call the resulting calculus *Network Conscious π-calculus* (NCPi). We consider networks without hierarchies (e.g. administrative domains), where some parts may be private to a process and the public part is shared, as in CHARM [12]. Networks can be used by many processes at the same time, but we impose some restrictions on how resources can be accessed. The calculus has the following features:

- We distinguish 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 between a and b. The syntax has new primitives for handling links and, since it is no more required for processes to communicate on shared channels, an extended output primitive is introduced that specifies not only the emission site but also the destination one.
- We provide two semantics: an interleaving one, inspired by the π-calculus early semantics, where an observation is a sequence of links representing the observable part of a routing path, and a concurrent one, where concurrent transmissions can be observed in the form of a multiset of paths.
- The behavioural equivalence in the interleaving case is not preserved by the input prefix, as in the π-calculus, but in the concurrent case it is preserved by all operators, hence it is a *congruence*. This is because the concurrent semantics provides more observations than the interleaving one, resulting in a finer, compositional bisimilarity. A first evidence of this fact is the classical counterexample not applying in the concurrent case: $\overline{a} \mid b$ and $\overline{a}.b + b.\overline{a}$, which are bisimilar according to the interleaving semantics, are distinguished, because (using an intuitive syntax) $\overline{a} \mid b \xrightarrow{\overline{a} \mid b} \mathbf{0}$ while $\overline{a}.b + b.\overline{a} \not\xrightarrow{\overline{a} \mid b}$.

We choose to have labelled connectors, instead of anonymous ones as in [8] and [10], for two main reasons. First of all, they are intended to model transportation services with distinct features (cost, bandwidth ...), which could be encoded in the label type, as we already do for the connectors' source and target. In any case, NCPi allows one to recover anonymous connectors through the restriction operator. Second, this allows us to reuse most of the notions of the π-calculus (renaming, α-conversion, extrusion ...), suitably extended.

The main result of our paper is that bisimilarity on the concurrent semantics is a congruence. This is a desirable property for a process calculus, because it allows for the compositional analysis of systems. The authors of [8] and [10] follow another approach to compositionality: they start from a reduction semantics, guess a suitable notion of barb, define barbed congruence by closing w.r.t. all the contexts, and then characterize it as a bisimulation equivalence on a labelled version of the transition system. In general, this approach leads to a labelled semantics with very succint observations, but may resort to non-standard notions of bisimilarity, where the closure under contexts is "hardwired". Instead, we show that we can gain the congruence property through a concurrent extension of the semantics while keeping the notion of bisimilarity as standard as possible. We emphasize that interleaving semantics is far from being natural in

this distributed setting. In fact, it is based on a mutual exclusion mechanism between remote actions which is simpler from a formal point of view, but not realistic for modelling concurrent systems.

Bisimilarity not being a congruence for the π-calculus depends on the interleaving nature of the semantics, and not on the language itself. In fact, we will show that, if we equip π-calculus with a concurrent semantics, the congruence property holds. This has already been shown in [13, 14], but the semantics presented there allows observing the channel where a synchronization is performed, whereas our concurrent semantics is closer to the π-calculus, in the sense that we adopt a synchronization mechanism that hides such channel.

Synopsis. In section 2 we show a motivating example. In section 3 we present the syntax of the language. In section 4 we describe its interleaving semantics. In section 5 we model a simple routing protocol in the interleaving setting. In section 6 we present the concurrent semantics and we show that its bisimilarity is a congruence. An informal proof of the latter property is in the appendix.

2 Motivating example

We consider the system made of a network manager M, using a reserved site m, and two processes p and q, which access the network respectively through the sites a and b. The manager is the only entity that can create new links and grant access to them. The process p wants to send a message to q, but we assume that there are no links between a and b allowing p and q to communicate. The processes are

$$\begin{array}{ll} M = m(x).m(y).(l_{xy})(\overline{m}xl_{xy}.M) & q = b(x).q' \\ p \;\; = \overline{a}ma.\overline{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \overline{a}bc.p') & L(l_{xy}) = l_{xy}.L(l_{xy}) \end{array}$$

M receives two sites at m, creates a new link between them and sends this link from m to the first of the received sites. The process p sends a and b from a to m, waits for a link at a and then evolves to the parallel composition of two components: the first component activates a transportation service over the received link, which can be used by the other component; the second component sends c from a to b. The process q simply waits for a datum at b. Finally, the process L repeatedly activates a transportation service over its argument: this is necessary, because transportation services can only be used once. The whole system is $S = p \mid M \mid q \mid L(l_{am}) \mid L(l'_{ma})$, where l_{am} and l'_{ma} are the links that p and M use to interact.

We have that p, $L(l_{am})$ and M can do the following transitions

$$\begin{array}{l} p \xrightarrow{\bullet;\overline{a}ma} \overline{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \overline{a}bc.p') \\ L(l_{am}) \xrightarrow{a;l_{am};m} L(l_{am}) \\ M \xrightarrow{mma;\bullet} m(y).(l_{ay})\overline{m}al_{ay}.M \end{array}$$

where $\bullet; \overline{a}ma$ represents the beginning of transmission as a path of length zero, analogous to the π-calculus output action: the $\bullet$ on the left side indicates that the path can only extend rightward, i.e. subsequent hops will be listed after $\bullet$ from left to right in the form of a sequence of links; the string $\overline{a}ma$ describes the path, telling (from left to right) the site where the datum is available, the destination site and the datum. Symmetrically, $mma; \bullet$ means that a, which has destination m, is received at m and then goes through a path of length zero; it is analogous to the π-calculus input action. The label $a; l_{am}; m$ represents the activation of a transportation service over l_{am}.

When these processes are put in parallel in S, the above paths can be concatenated, resulting in a path that represents a complete transmission over l_{am}

$$S \xrightarrow{\bullet; l_{am}; \bullet} \overline{a}mb.a(l_{(xy)}).(L(l_{xy}) \mid \overline{a}bc.p') \mid m(y).(l_{ay})(\overline{m}al_{ay}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) \quad .$$

As in the π-calculus, the transmitted datum, namely a, is not observable. Then, a sequence of possible transitions after this one is:

$$\begin{aligned}
\cdots &\xrightarrow{\bullet; l_{am}; \bullet} a(l_{(xy)}).(L(l_{xy}) \mid \overline{a}bc.p') \mid (l_{ab})(\overline{m}al_{ab}.M) \mid q \mid L(l_{am}) \mid L(l'_{ma}) \\
&\qquad\qquad\qquad

Definition 1 (NCPi processes). *The NCPi processes are defined as follows:*

$$p ::= \mathbf{0} \mid \pi.p \mid p+p \mid p \,|\, p \mid (r)p \mid A(r_1, r_2, \ldots, r_n)$$
$$r ::= a \mid l_{ab} \qquad s ::= a \mid l_{(ab)} \qquad \pi ::= \overline{a}br \mid a(s) \mid l_{ab} \mid \tau$$

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

where $a, b \in \mathcal{S}, l_{ab} \in \mathcal{L}$.

We have the usual inert process, nondeterministic choice and parallel composition. For the recursive definition, we require that formal parameters do not have names in common, because otherwise we might have type dependencies between parameters, e.g. in $A(a, l_{(ab)})$ one of the second parameter's endpoints depends on the first parameter. Prefixes can have the following forms:

- The *output prefix* $\overline{a}br$: $\overline{a}br.p$ can send the datum r from a addressed to b and continue as p. Notice that, unlike π-calculus, the destination site can be different than the emission one.
- The *input prefix* $a(s)$: $a(s).p$ can receive at a a datum to be bound to s and continue as p. The intended meaning of $c(l_{(ab)}).p$ is an atomic, polyadic version of $c(a).c(b).c(l_{ab}).p$. Here a monadic link input prefix $c(l_{ab}).p$ is not allowed, since it would introduce a matching capability we prefer not to provide. Consequently, a and b are not free in $c(l_{(ab)}).p$.
- The τ *prefix*: $\tau.p$ can perform an internal action and continue as p.
- The *link prefix* l_{ab}: $l_{ab}.p$ can offer to the environment the service of transporting a datum from a to b through l and then continue as p.

Finally, we have the *restriction* (r): r is private in $(r)p$, i.e. it cannot be observed as free name in a communication. Notice that a and b are free in $(l_{ab})p$. Sequences of restrictions will be denoted by capital letters (R) and will be manipulated using set operations.

We define the set $\mathrm{fn}(p)$ of free names in p as:

$$\begin{aligned}
\mathrm{fn}(\mathbf{0}) &= \varnothing & \mathrm{fn}(\tau.p) &= \mathrm{fn}(p) \\
\mathrm{fn}(\overline{a}br.p) &= \{a, b\} \cup \mathrm{n}(r) \cup \mathrm{fn}(p) & \mathrm{fn}(l_{ab}.p) &= \{l_{ab}, a, b\} \cup \mathrm{fn}(p) \\
\mathrm{fn}(b(a).p) &= \{b\} \cup (\mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a)) & \mathrm{fn}(a(l_{(bc)}).p) &= \{a\} \cup \mathrm{fn}(p) \setminus (\{b, c\} \cup \mathcal{L}_b \cup \mathcal{L}_c) \\
\mathrm{fn}((a)p) &= \mathrm{fn}(p) \setminus (\{a\} \cup \mathcal{L}_a) & \mathrm{fn}((l_{ab})p) &= \{a, b\} \cup \mathrm{fn}(p) \setminus \{l_{ab}\} \\
\mathrm{fn}(p+q) &= \mathrm{fn}(p \,|\, q) = \mathrm{fn}(p) \cup \mathrm{fn}(q) & \mathrm{fn}(A(r_1, \ldots, r_n)) &= \mathrm{n}(r_1) \cup \cdots \cup \mathrm{n}(r_n)
\end{aligned}$$

where $A(s_1, \ldots, s_n) \stackrel{\text{def}}{=} p$ implies $\mathrm{fn}(p) \subseteq \mathrm{n}(s_1) \cup \cdots \cup \mathrm{n}(s_n)$. Notice the definition of $\mathrm{fn}((a)p)$: if a link having a as one of its endpoints appears in p, then it is considered bound. Similarly for $b(a).p$ and $a(l_{(bc)}).p$. This intuitively means that a global link cannot have private endpoints, analogously to what happens for free processes in a well-formed state of a CHARM [12]: their variables must belong to the global part.

The notion of renaming is defined as follows.

α-equivalence

$$(a)p \equiv (a')p[^{a'}/_{a}] \qquad b(a).p \equiv b(a').p[^{a'}/_{a}] \qquad a' \notin \mathrm{fn}((a)p)$$

$$(l_{ab})p \equiv (l'_{ab})p[^{l'_{ab}}/_{l_{ab}}] \qquad \forall a', b' : l'_{a'b'} \notin \mathrm{fn}((l_{ab})p)$$

$$a(l_{(bc)}).p \equiv a(l'_{(b'c')}).p[^{l'_{b'c'}}/_{l_{(bc)}}] \qquad b', c' \notin \mathrm{fn}(a(l_{(bc)}).p) \wedge \forall b'', c'' : l'_{b''c''} \notin \mathrm{fn}(a(l_{(bc)}).p)$$

Unfolding law

$$A(r_1, \ldots, r_n) \equiv p[^{r_1}/_{s_1}, \ldots, ^{r_n}/_{s_n}] \quad \text{if } A(s_1, \ldots, s_n) \stackrel{\text{def}}{=} p$$

Fig. 1: Structural congruence axioms for well-formed processes.

Definition 2 (Renaming). *A renaming σ is a pair of functions $\langle \sigma_{\mathcal{S}} : \mathcal{S} \to \mathcal{S}, \sigma_{\mathcal{L}} : \mathcal{L} \to \mathcal{L} \rangle$ such that $\sigma_{\mathcal{L}}(l_{ab}) = l'_{a'b'}$ implies $\sigma_{\mathcal{S}}(a) = a'$ and $\sigma_{\mathcal{S}}(b) = b'$. We denote by $r\sigma$ the result of applying the appropriate component of σ to r.*

The condition relating $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{L}}$ ensures that σ acts as a graph homomorphism, i.e. each link is renamed by $\sigma_{\mathcal{L}}$ to a link whose endpoints are the image through $\sigma_{\mathcal{S}}$ of the original link's endpoints. Some notation: we write $[^{r'_1}/_{r_1}, ^{r'_2}/_{r_2}, \ldots, ^{r'_n}/_{r_n}]$ to indicate the function mapping r_1 to r'_1, r_2 to r'_2 ... r_n to r'_n, and we write $[^{l'_{a'b'}}/_{l_{(ab)}}]$ as a shorthand for $[^{a'}/_{a}, ^{b'}/_{b}, ^{l'_{a'b'}}/_{l_{ab}}]$. Notice that $[^{a'}/_{a}]$ does not uniquely characterize a renaming. In fact, while surely $\overline{a}bc[^{a'}/_{a}] = \overline{a'}bc$, $a \notin \{b, c\}$, for $l_{ab}[^{a'}/_{a}]$ we only know that it must belong to $\mathcal{L}_{a'b}$. Thus we should avoid applying such renaming to a link l_{ab}, since the result would be undefined. A special case (see below) is when l_{ab} is bound.

Now we introduce *well-formed* NCPi processes. Informally, a process is well-formed if each bound link it contains is bound explicitely, and not as a side-effect of binding a site, and if two links with the same label but different endpoints do not appear free in any of its suprocesses. For instance: $a(b).l_{bc}.p$ and $(l_{ab})l_{ab}.l_{cd}.p$ are not well-formed: the former because l_{bc} is implicitly bound by $a(b)$, the latter because l labels two links between different sites.

Definition 3 (Well-formed NCPi processes). *A NCPi process p is* well-formed *if for every subterm q:*

(i) $q = (a)p'$ *implies* $\mathrm{fn}(q) = \mathrm{fn}(p') \setminus \{a\}$*;*
(ii) $q = b(a).p'$ *implies* $\mathrm{fn}(q) = \{b\} \cup \mathrm{fn}(p') \setminus \{a\}$*;*
(iii) $q = c(l_{(ab)}).p'$ *implies* $\mathrm{fn}(q) = \{c\} \cup \mathrm{fn}(p') \setminus \{a, b, l_{ab}\}$*;*
(iv) $l_{ab}, l'_{cd} \in \mathrm{fn}(q)$ *and* $ab \neq cd$ *implies* $l \neq l'$*.*

A first consequence of this definition is that we do not need to subtract $\mathcal{L}_a$ in $\mathrm{fn}(b(a).p)$ and $\mathrm{fn}((a)p)$, and $\mathcal{L}_b$, $\mathcal{L}_c$ in $\mathrm{fn}(a(l_{(bc)})).p$ if p is well-formed.

Well-formedness also allows us to say how a generic substitution can act on processes as a proper renaming. This is needed in order to define α-conversion,

which in fact is given in Fig. 1 for well-formed processes only. α-conversion for a restricted process is simply $(a)p \equiv (a')p[^{a'}/_{a}]$, with $a' \notin \mathrm{fn}((a)p)$, where $[^{a'}/_{a}]$ is never applied to a link l_{ab}, since such link cannot be free in p. If it is bound, i.e. if $(l_{ab})p'$ is a subprocess of p, then we simply have inductively $((l_{ab})p')[^{a'}/_{a}] \equiv (l'_{a'b})(p'[^{l'_{ab}}/_{l_{ab}}])[^{a'}/_{a}]$, with $l'_{ab}[^{a'}/_{a}] = l'_{a'b}$, for any l'_{ab} that can replace l_{ab} through α-conversion; this preserves property (iv) of well-formedness, since this property for $l'_{a'b}$ is inherited from the same property for l'_{ab}. Moreover, in order to mantain this property, capture must be avoided not only in the presence of $l'_{ab} \in \mathrm{fn}(p)$, but also of $l'_{a'b'} \in \mathrm{fn}(p)$, for every a' and b'. A similar restriction holds also when α-converting $a(l_{(bc)}).p$. Notice that we can α-convert it also with respect to b,c or l_{bc} separately. In the following we will consider only well-formed processes.

4 Interleaving Semantics

A *path*, denoted by α, represents the observable part of the routing path of a single datum. Paths are the observations for the interleaving semantics.

Definition 4 (Paths). *Paths are defined as follows:*

$$\begin{array}{l} \alpha ::= a; W; b \ \mid \ \bullet; W; \bullet \ \mid \ \bullet; W; \overline{a}br \ \mid \ abr; W; \bullet \\ \quad \mid \ ab(s); W; \bullet \ \mid \ (r)\alpha \qquad \mathrm{n}(s) \cap (\mathrm{n}(W) \cup \{a, b\}) = \varnothing \\ r ::= a \ \mid \ l_{ab} \qquad s ::= a \ \mid \ l_{(ab)} \qquad W ::= l_{ab} \ \mid \ W; W \ \mid \ \epsilon \end{array}$$

where $a, b \in \mathcal{S}$ and $l_{ab} \in \mathcal{L}$. The structural congruence $\equiv_\alpha$ is given by the monoidal axioms for strings, where ; and ϵ are the multiplication and the identity, and by $(r)(r')\alpha \equiv_\alpha (r')(r)\alpha$, $\mathrm{n}(r) \cap \mathrm{n}(r') = \varnothing$.

A path can be: a *service path* $a; W; b$, representing a transportation service from a to b that employs the resources listed in W and possibly other private, unobservable resources, or a sequence starting and/or ending with $\bullet$, which represents an actual transmission over W. In the latter case, it can be:

- a *free output path*, if $\overline{a}br$ is on the right, representing the emission of r, whose destination is b, at a;
- an *input path*, if abr or $ab(s)$ is on the left. In the former case, it is called *free input path* and means that r, whose destination is b, is received at a; in the latter case, it is called *bound input path* and s is a placeholder for the received name;
- a *complete path*, if $\bullet$ is on both sides, meaning that the transmission has already been completed;
- an *extrusion path* if it is of the form $(r)\alpha$, meaning that r is being extruded through α.

We will use W_α to denote the sequence of links of α and $|W_\alpha|$ to denote the set of links appearing in W_α. The set of free names $\mathrm{fn}(\alpha)$ of a free path α is $n(\alpha)$ and the set of its bound names $\mathrm{bn}(\alpha)$ is empty; for the other paths we have

$$\mathrm{fn}(ab(s); W; \bullet) = \{a, b\} \cup \mathrm{n}(W) \quad \mathrm{fn}((a)\alpha) = \mathrm{fn}(\alpha) \setminus (\{a\} \cup \mathcal{L}_a) \quad \mathrm{fn}((l_{ab})\alpha) = \mathrm{fn}(\alpha) \setminus \{l_{ab}\}$$

and $\mathrm{bn}(\alpha) = \mathrm{n}(\alpha) \setminus \mathrm{fn}(\alpha)$. The *objects* $\mathrm{obj}(\alpha)$ are

$$\mathrm{obj}(abr; W; \bullet) = \mathrm{obj}(\bullet; W; \overline{a}br) = \{b\} \cup \mathrm{n}(r) \quad \mathrm{obj}((a)\alpha) = \mathrm{obj}(\alpha) \setminus (\{a\} \cup \mathcal{L}_a)$$
$$\mathrm{obj}(a; W; b) = \mathrm{obj}(\bullet; W; \bullet) = \varnothing \quad \mathrm{obj}((l_{ab})\alpha) = \mathrm{obj}(\alpha) \setminus \{l_{ab}\} \quad \mathrm{obj}(ab(s); W; \bullet) = \{b\}$$

which correspond to the free objects in the π-calculus. We write $\mathrm{obj}_{\mathrm{in}}(\alpha)$ for $\mathrm{n}(r)$ when $\alpha = (R)\ abr; W; \bullet$. We call *interaction sites* of α, denoted by $\mathrm{is}(\alpha)$, those sites in α where an interaction with other processes may happen, namely α's endpoints if α is a service path and the emission or reception site if it is an input or output path. These correspond to subjects of the π-calculus.

Anologously to what happens for processes, in $(a)\alpha$ the restriction may implicitely bind some elements of $\mathcal{L}_a$ occurring free in α. This does not happen for *well-formed paths*.

Definition 5 (Well-formed paths). *A path α is well-formed if $\alpha = (r)\alpha'$ implies* $\mathrm{fn}(\alpha) = \mathrm{fn}(\alpha') \setminus \{r\}$.

Now we introduce the *hiding* operation, which we will use in the SOS rules to implement the effects that restricting a name of a process has on its paths.

Definition 6 (Hiding operation). *The hiding operation / acts on sequences of links as follows*

$$\epsilon/r = \epsilon \qquad (W; W')/r = (W/r); (W'/r) \qquad l_{ab}/r = \begin{cases} \epsilon & r \in \{a, b, l_{ab}\} \\ l_{ab} & \textit{otherwise} \end{cases}$$

Its extension to paths α/r is obtained by replacing W_α with W_α/r in α.

This operation simply removes each occurrence of the no longer observable site or link from its argument.

We will require that paths inferred through the rules satisfy the following property.

Definition 7 (Simple path). *A path α is* simple *if $W_\alpha = W_1; W_2$ implies $|W_1| \cap |W_2| = \varnothing$, for any W_1 and W_2.*

A path is simple if it does not use the same link twice, e.g. $\bullet; l_{ab}; l'_{bc}; l_{ab}; \overline{b}dr$ is not simple. By restricting to simple paths we formalize the fact that links stand for consumable resources, so they must be employed in a mutually exclusive way during a communication. We can now define the *NCPi interleaving transition system.*

Definition 8 (NCPi interleaving transition system). *Consider the rules in Fig.2, where observations are up to $\equiv_\alpha$. The NCPi transition system is the smallest transition system obtained from these rules and closed under the equivalence relation $\equiv_I$ obtained by extending $\equiv$ with the commutative monoidal axioms for $|$. Explicitely: if $p \xrightarrow{\alpha} q$, $p \equiv_I p'$ and $q \equiv_I q'$ then $p' \xrightarrow{\alpha} q'$.*

(FREE-INPUT)	$a(s).p \xrightarrow{aar;\bullet} p[r/s]$	$r{=}l_{ab} \Rightarrow \forall a'b' : l_{a'b'} \notin \mathrm{fn}(a(s).p)$
(BOUND-INPUT)	$a(s).p \xrightarrow{aa(s);\bullet} p$	
(OUTPUT)	$\overline{a}br.p \xrightarrow{\bullet;\overline{a}br} p$	
(LINK)	$l_{ab}.p \xrightarrow{a;l_{ab};b} p$	$a \neq b$
(INTERNAL)	$\tau.p \xrightarrow{\bullet;\bullet} p$	
(RES)	$\dfrac{p \xrightarrow{\alpha} q}{(r)p \xrightarrow{\alpha/r} (r)q}$	$r \notin \mathrm{obj}(\alpha) \cup \mathrm{bn}(\alpha) \cup \mathrm{is}(\alpha)$
(OPEN)	$\dfrac{p \xrightarrow{\alpha} q}{(r)p \xrightarrow{(r)(\alpha/r)} q}$	$r \in \mathrm{obj}(\alpha) \setminus (\mathrm{is}(\alpha) \cup \mathrm{obj}_{\mathrm{in}}(\alpha))$
(SUM-L)	$\dfrac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'}$	
(PAR)	$\dfrac{p_1 \xrightarrow{\alpha} q_1}{p_1 \mid p_2 \xrightarrow{\alpha} q_1 \mid p_2}$	$\mathrm{bn}(\alpha) \cap \mathrm{fn}(p_2) = \varnothing$ $l_{ab} \in \mathrm{bn}(\alpha) \cup \mathrm{obj}_{\mathrm{in}}(\alpha) \Rightarrow$ $\forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(p_2)$
(COM)	$\dfrac{p_1 \xrightarrow{(R)\ \bullet;W;\overline{a}br} q_1 \quad p_2 \xrightarrow{(R')\ abx;W';\bullet} q_2}{p_1 \mid p_2 \xrightarrow{\bullet;W;W';\bullet} (R)\ (q_1 \mid q_2\sigma)}$	$b \in R \iff b \in R'$ $x, \sigma = \begin{cases} [r/s], (s) & r \in R \\ id, r & r \notin R \end{cases}$ $l_{ab} \in R \Rightarrow$ $\forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(p_2)$
(SRV-OUT)	$\dfrac{p_1 \xrightarrow{(R)\ \bullet;W;\overline{a}br} q_1 \quad p_2 \xrightarrow{a;W';c} q_2}{p_1 \mid p_2 \xrightarrow{(R)\ \bullet;W;W';\overline{c}br} q_1 \mid q_2}$	$R \cap \mathrm{fn}(p_2) = \varnothing$ $l_{ab} \in R \Rightarrow$ $\forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(p_2)$
(SRV-IN)	$\dfrac{p_1 \xrightarrow{a;W;b} q_1 \quad p_2 \xrightarrow{(R)\ bcx;W';\bullet} q_2}{p_1 \mid p_2 \xrightarrow{(R)\ acx;W;W';\bullet} q_1 \mid q_2}$	$(R \cup \mathrm{bn}(x)) \cap \mathrm{fn}(p_1) = \varnothing$ $l_{ab} \in \mathrm{n}(x) \Rightarrow$ $\forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(p_1)$
(SRV-SRV)	$\dfrac{p_1 \xrightarrow{a;W;b} q_1 \quad p_2 \xrightarrow{b;W';c} q_2}{p_1 \mid p_2 \xrightarrow{a;W;W';c} q_1 \mid q_2}$	

The paths inferred by (COM) and (SRV-*) must be simple

Fig. 2: NCPi interleaving operational rules. The rule (SUM-L) also has a symmetric version.

The rules (FREE-INPUT) and (BOUND-INPUT) treat the reception of a global and a private name, respectively, while (OUTPUT) treats the emission of a global name. These actions are represented as paths of length zero. As in the early π-calculus, a renaming must be applied to the continuation in the free input case; by well-formedness, such renaming can always be extended to act as a proper graph homomorphism. The reception of a global link should be treated carefully: the rule forbids it whenever another link with the same label, but different endpoints, already occurs free in the process, because the renaming would break well-formedness.

The rule (LINK) is used to provide a transportation service to the environment, but we forbid services from a site to itself.

The rule (INTERNAL) infers a transition labelled with the empty path $\bullet;\bullet$, representing an internal action.

The rule (RES) infers a transition of $(r)p$ from the transitions of p, but it considers only those transitions such that r is not an interaction site and is not sent or received. This side condition reflects that of the π-calculus rule, where r must not be the subject or the object of the premise's action, and its purpose is to avoid captures: e.g. if $(a)b(c).p$ is such that $c \in \mathrm{fn}(p)$ and it is allowed to perform $bba;\bullet$, then a would be captured in the continuation $(a)p[^a/_c]$.

The rule (OPEN) infers the scope extrusion of a restricted name r by turning a path of p into an extrusion path of $(r)p$, provided that r is an object in the original path, but not the datum of an input or an interaction site. Notice that the rule allows us to "extrude" the destination site: the intuition is that we can use global resources to send or receive a datum to/from a local site, which becomes global if the communication is not complete.

The rule (SUM-L) is an obvious extension of the corresponding π-calculus rule.

The rule (PAR) is similar to the π-calculus one, with the usual side condition on the label's fresh names, but in addition we require that, whenever α represents the reception or the extrusion of a link, no other links with the same label but different endpoints occur free in p_2, because otherwise the continuation may not be well-formed.

The rules (COM) is used to complete a transmission, covering both free and bound names communication. In the case of bound names communication, a renaming is applied and the extruded names become restricted in the continuation. The side conditions ensure that input and output paths always have the same destination site, even when this is bound, and that received links do not break well-formedness.

The rules (SRV-IN) and (SRV-OUT) state that a process can use a transportation service, provided by another process running in parallel, if their paths meet, the bound names transmitted are fresh with respect to the service provider and link labels in the resulting transition do not break well-formedness. The rule (SRV-SRV) is used to compose two services.

The transition system satisfies the following property.

Proposition 1. *If $p \xrightarrow{\alpha} q$ then α is simple and well-formed, and q is well-formed.*

Now we introduce the interleaving behavioural equivalence.

Definition 9 (Interleaving network conscious bisimilarity). *A binary, symmetric and reflexive relation $\mathcal{R}$ is an* interleaving network conscious bisimulation *if $(p, q) \in \mathcal{R}$ and $p \xrightarrow{\alpha} p'$, with:*

(i) $\mathrm{bn}(\alpha) \cap \mathrm{fn}(q) = \varnothing$*;*
(ii) $l_{ab} \in \mathrm{bn}(\alpha) \cup \mathrm{obj}_{\mathrm{in}}(\alpha) \Rightarrow \forall a'b' \neq ab : l_{a'b'} \notin \mathrm{fn}(q)$

implies that there is q' such that $q \xrightarrow{\alpha} q'$ and $(p', q') \in \mathcal{R}$. The bisimilarity is the largest such relation and is denoted by $\sim_I^{NC}$.

Condition i is standard, while ii rules out the transitions of p that q may not be able to simulate due to well-formedness. Notice that a consequence of defining the semantics up to structural congruence is that $\equiv_I \subseteq \sim_I^{NC}$.

Theorem 1. *$\sim_I^{NC}$ is a congruence w.r.t all NCPi operators except the input prefix.*

Finally, we can establish a relation between a subcalculus of the interleaving NCPi and the π-calculus.

Proposition 2. *Let* linkless NCPi *be the subcalculus of NCPi such that no links appear in processes and the output prefix is of the form $\overline{a}ab$. Then there is a one-to-one correspondence between processes and transitions of π-calculus and linkless NCPi.*

This enconding maps $\overline{a}b$ to $\overline{a}ab$ or $\bullet; \overline{a}ab$, depending on whether it is used as prefix or as action; the other cases are obvious. By homomorphic extension we get the encoding for processes and transitions, which is one-to-one because the rules in Fig.2, if we restrict syntax as described, can be expressed in terms of simpler rules matching the π-calculus ones. This proposition explains why $\sim_I^{NC}$ is not a congruence.

5 Example: routing protocol

In this section we model a simple routing protocol, similar to BGP [15]. This protocol assumes that the network is composed of disjoint groups of networks, each referring to a single administrative authority, called *Autonomous Systems* (AS). Some of the ASs' routers act as *gateways* between the AS they belong to and other networks. The protocol takes care of the routing mechanism between ASs in a distributed manner: each gateway has a *routing table*, filled by the protocol, whose entries specify which is the next hop along the "best" path towards some destination; this information will be used to forward the incoming data.

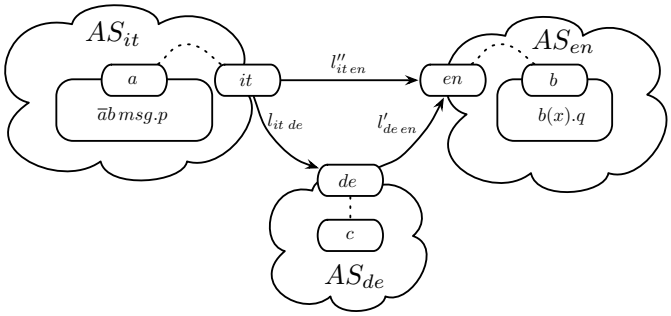

Fig. 3: Example network.

In our model, both routers and hosts are represented as sites, and network connections are represented as links. Autonomous systems are generic processes whose links are all restricted, because these links represent local services. Routing tables are modelled as functions RT_g, one for each gateway g, such that $RT_g(a)$ is a link l_{gh} to some other gateway h, representing the next hop of the best path towards a. The forwarding is implemented at the SOS level by employing the following rule for gateways

$$(\text{ROUTE})\ \frac{p_1 \xrightarrow{(R)\bullet;W;\overline{g}ar} p_1' \qquad p_2 \xrightarrow{g;l_{gh};h} p_2' \qquad RT_g(a) = l_{gh}}{p_1 \,|\, p_2 \xrightarrow{(R)\bullet;W;l_{gh};\overline{h}ar} p_1' \,|\, p_2'}$$

Now, consider the network depicted in Fig.3. We have three ASs: an Italian one, a German one and an English one, whose gateways are respectively the sites it, de and en; and we have two processes willing to communicate from site a in AS_{it} to site b in AS_{en}[1]. Suppose there is a path from a to it in AS_{it}, the routing tables are such that $RT_{it}(b) = l_{it\,de}$ and $RT_{de}(b) = l'_{de\,en}$, and that there is a path in AS_{en} from en to b. Let C denote $L(l_{it\,de}) \,|\, L(l'_{de\,en}) \,|\, L(l''_{it\,en})$, where $L(l_{(xy)}) \stackrel{\text{def}}{=} l_{xy}.L(l_{xy})$, modelling the connections between the ASs. Then (ROUTE) yields

$$AS_{it} \,|\, AS_{en} \,|\, AS_{de} \,|\, C \xrightarrow{\bullet;l_{it\,de};l'_{de\,en};\bullet} AS'_{it} \,|\, AS'_{en} \,|\, AS_{de} \,|\, C \ .$$

Notice that only the part of the path between the gateways is observable.

6 Concurrent semantics

Interleaving semantics can be considered inadequate for distributed system with partially asynchronous behaviour, since it implicitely assumes the existence of a

[1] For the sake of brevity, the roles played by sites, such as "gateway of a given AS", are stated informally here, but they could be formalized through a type system.

Commutative monoid laws for |	**Scope extension laws**	
$\Lambda_1 \mid \Lambda_2 \equiv_\Lambda \Lambda_2 \mid \Lambda_1$	$(r)(r')\Lambda \equiv_\Lambda (r')(r)\Lambda$	$r \notin \mathrm{n}(r'), r' \notin \mathrm{n}(r)$
$(\Lambda_1 \mid \Lambda_2) \mid \Lambda_3 \equiv_\Lambda \Lambda_1 \mid (\Lambda_2 \mid \Lambda_3)$	$\Lambda_1 \mid (r)\Lambda_2 \equiv_\Lambda (r)(\Lambda_1 \mid \Lambda_2)$	$r \notin \mathrm{Fn}(\Lambda_1)$
$\Lambda \mid \mathbf{1} \equiv_\Lambda \mathbf{1} \mid \Lambda \equiv_\Lambda \Lambda$		
Singleton concurrent path		
$\beta_1 \equiv_\alpha \beta_2 \Rightarrow \beta_1 \equiv_\Lambda \beta_2$		

Fig. 4: Structural congruence of concurrent paths.

central arbiter who grants access to resources. This criticism is particularly relevant for our network-conscious calculus. Here we present a concurrent semantics where we can observe multisets of paths covered at the same time, instead of single paths. These are denoted by Λ and called *concurrent paths*.

Definition 10 (Concurrent paths). *Concurrent paths are defined as follows:*

$$\Lambda ::= \mathbf{1} \ \mid \ \beta \ \mid \ \Lambda_1 | \Lambda_2 \ \mid \ (r)\Lambda$$

where β is a path without extrusion restrictions (see definition 4).

Concurrent paths can have the following forms:

- The *empty concurrent path* $\mathbf{1}$ indicates that no activity is performed.
- The *singleton concurrent path* β is a concurrent path made of a single path.
- The *union* $\Lambda_1 \mid \Lambda_2$ means that the paths in Λ_1 and Λ_2 are being traversed *at the same time*.
- The *extrusion restriction* $(r)\Lambda$ indicates that r is being extruded through one or more paths in Λ.

The free names $\mathrm{Fn}(\Lambda)$ and bound names $\mathrm{Bn}(\Lambda)$ of a concurrent path Λ are defined by an obvious induction on the syntax, where the base cases are $\mathrm{fn}(\beta)$ and $\mathrm{bn}(\beta)$; as usual, we have to be careful with the following cases

$$\mathrm{Fn}((a)\Lambda) = \mathrm{Fn}(\Lambda) \setminus (\{a\} \cup \mathcal{L}_a) \qquad \mathrm{Bn}((a)\Lambda) = \mathrm{Bn}(\Lambda) \cup \{a\} \cup (\mathcal{L}_a \cap \mathrm{n}(\Lambda))$$

The sets $\mathrm{Obj}(\Lambda)$, $\mathrm{Is}(\Lambda)$ and $\mathrm{Obj}_{\mathrm{in}}(\Lambda)$ are defined as $\mathrm{Fn}(\Lambda)$, considering respectively $\mathrm{obj}(\beta)$, $\mathrm{is}(\beta)$ and $\mathrm{obj}_{\mathrm{in}}(\beta)$ as base cases. In Fig.4 the axioms defining the structural congruence $\equiv_\Lambda$ for concurrent paths are shown: the operator $\mid$ defines a commutative monoid with $\mathbf{1}$ as identity, there are axioms for extending the scope of a restriction and finally an axiom saying that if two paths are structurally congruent, so are the singleton concurrent paths containing them.

The notion of well-formedness for concurrent paths is defined as follows.

Definition 11 (Well-formed concurrent paths). *A concurrent path Λ is* well-formed *if for every subterm Λ' of the form $(a)\Lambda''$ we have* $\mathrm{fn}(\Lambda') = \mathrm{fn}(\Lambda'') \setminus \{a\}$.

For instance, the concurrent path $(c)(\bullet; \overline{a}bc \mid a; l_{ac}; c)$ is not well-formed, since (c) implicitely binds l_{ac}.

We specify a canonical form for concurrent paths.

Definition 12 (Concurrent paths in canonical form). *A concurrent path Λ is in* canonical form *if it has the form $(R)\Theta$, where R is a sequence of restrictions and Θ does not contain extrusion restrictions.*

Notice that binders of the form $ab(s)$ are still allowed in Θ.

Now we extend the hiding operation and the notion of simple path to concurrent paths.

Definition 13 (Hiding operation). *We denote by $/\!\!/$ the extension of $/$ to concurrent paths:*

$$\mathbf{1}/\!\!/r = \mathbf{1} \quad (\Lambda_1 \mid \Lambda_2)/\!\!/r = (\Lambda_1/\!\!/r) \mid (\Lambda_2/\!\!/r) \quad \beta/\!\!/r = \beta/r \quad ((r')\Lambda)/\!\!/r = \begin{cases} (r')(\Lambda/\!\!/r) & \text{if } r \neq r' \\ (r')\Lambda & \text{otherwise} \end{cases}$$

Definition 14 (Simple concurrent path). *A concurrent path is* simple *if each $\beta \in \Lambda$ is simple and, for any other $\beta' \in \Lambda$, $|W_\beta| \cap |W_{\beta'}| = \varnothing$.*

Here, besides the simplicity of all the paths in Λ, we also require that these paths do not share links.

The *NCPi concurrent transition system* is defined as follows.

Definition 15 (NCPi concurrent transition system). *Consider the rules in Fig.5, where observations are up to $\equiv_\Lambda$. The concurrent NCPi transition system is the smallest transition system obtained from these rules and closed under $\equiv$.*

The rules (RES) and (OPEN) are extensions of the corresponding interleaving rules.

The rule (IDLE) infers a "no-op" transition, enabling the parallel composition of processes to behave in an interleaving style.

The rule (PAR) subsumes its interleaving counterpart, because it implements the interleaving behaviour together with (IDLE). It also gives rise to the concurrent behaviour by making the union of the paths in the premise, while (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) take care of concatenating them, without modifying the source process.

This union is allowed only if the resulting concurrent path is simple, if we do not lose well-formedness due to inconsistent link labels, and if the concurrent path of each premise has bound names which are fresh w.r.t the other process and distinct from all the names occurring in the other concurrent path. This last condition avoids inferring transitions where the extruded name is free in the receiving process's continuation even if it has not been actually received, which might cause incorrect behaviours. For instance, consider the processes $p = (b)\overline{a}ab.b(c).p'$ and $q = a(d).\overline{d}de.q'$, and suppose $p \mid q \xrightarrow{(b)\bullet;\overline{a}ab \mid aab;\bullet} b(c).p' \mid \overline{b}be.q'[^b/_d]$ is allowed;

(RES) $$\frac{p \overset{\Lambda}{\Rightarrow} q}{(r)p \overset{\Lambda /\!\!/ r}{\Longrightarrow} (r)q} \qquad r \notin \mathrm{Obj}(\Lambda) \cup \mathrm{Bn}(\Lambda) \cup \mathrm{Is}(\Lambda)$$

(OPEN) $$\frac{p \overset{\Lambda}{\Rightarrow} q}{(r)p \overset{(r)(\Lambda /\!\!/ r)}{\Longrightarrow} q} \qquad r \in \mathrm{Obj}(\Lambda) \setminus (\mathrm{Is}(\Lambda) \cup \mathrm{Obj}_{\mathrm{in}}(\Lambda))$$

(IDLE) $$p \overset{\mathbf{1}}{\Rightarrow} p$$

(PAR) $$\frac{p_1 \overset{\Lambda_1}{\Longrightarrow} q_1 \quad p_2 \overset{\Lambda_2}{\Longrightarrow} q_2}{p_1 \mid p_2 \overset{\Lambda_1 \mid \Lambda_2}{\Longrightarrow} q_1 \mid q_2}$$

$\Lambda_1 \mid \Lambda_2$ is simple
$l_{ab} \in \mathrm{Bn}(\Lambda_i) \cup \mathrm{Obj}_{\mathrm{in}}(\Lambda_i) \Rightarrow$
$\quad \forall a'b' \neq ab : l_{a'b'} \notin \mathrm{n}(\Lambda_{3-i}) \cup \mathrm{fn}(p_{3-i})$
$\mathrm{Bn}(\Lambda_i) \cap (\mathrm{n}(\Lambda_{3-i}) \cup \mathrm{fn}(p_{3-i})) = \varnothing$

(COM) $$\frac{p \overset{(R)\ (\bullet;W;\overline{a}br \mid ab'x;W';\bullet \mid \Theta)}{\Longrightarrow} q}{p \overset{(R')\ (\bullet;W;W';\bullet \mid \Theta)}{\Longrightarrow} (R'')\ q(\sigma_b \circ \sigma_r)}$$

$R' = R \cap \mathrm{Obj}(\Theta)$
$R'' = (R \setminus R') \cap (\{b\} \cup \mathrm{n}(r))$
see tables (b) and (c)

(SRV-IN) $$\frac{p \overset{(R)\ (a;W;b \mid bcx;W';\bullet \mid \Theta)}{\Longrightarrow} q}{p \overset{(R)\ (acx;W;W';\bullet \mid \Theta)}{\Longrightarrow} q}$$

(SRV-OUT) $$\frac{p \overset{(R)\ (\bullet;W;\overline{a}br \mid a;W';c \mid \Theta)}{\Longrightarrow} q}{p \overset{(R)\ (\bullet;W;W';\overline{c}br \mid \Theta)}{\Longrightarrow} q}$$

(SRV-SRV) $$\frac{p \overset{a;W;b \mid b;W';c \mid \Lambda}{\Longrightarrow} q}{p \overset{a;W;W';c \mid \Lambda}{\Longrightarrow} q}$$

(a)

– $b' = b$ – $\sigma_b = id$	– $b, b' \in R$ – $\sigma_b = [b/b']$

(b)

– $r \notin R$ – $x = r$ – $\sigma_r = id$	– $r \in R$ – $x = (s)$ – $\sigma_r = [r/s]$

(c)

Fig. 5: NCPi concurrent operational rules: (a) shows the SOS rules (the omitted ones are in common with the interleaving semantics); (b) and (c) are the possible configurations for (COM). Any pair of configurations, one from (b) and one from (c), is valid (four possibilities).

now the two components of the continuation can synchronize on b even if its scope extension has not actually been accomplished, which is clearly incorrect.

Here it is simpler to give just one rule (COM) for all kinds of communications. In the case of a bound name communication, the rule's behaviour is quite different from the interleaving one: when two complementary paths are turned into a complete path, their extruded names must not necessarily be restricted in the continuation, because there may be other paths transporting them; another difference is that a renaming needs to be applied to the continuation if the destination sites are bound, because (PAR) side conditions do not allow for bound names equality.

The premises of (COM), (SRV-IN) and (SRV-OUT) must have their concurrent paths in canonical form: this is always possible, thanks to (PAR) side conditions. Notice that there is no need for the monoidality axioms of the parallel operator here, because $\equiv_\Lambda$ allows reordering the elements of concurrent paths.

We have the same result as the interleaving transition system.

Proposition 3. *If $p \overset{\Lambda}{\Rightarrow} q$ then Λ is simple and well-formed, and q is well-formed.*

The interleaving and concurrent transition systems are related as expected.

Theorem 2. *$p \xrightarrow{\alpha} q$ if and only if $p \overset{\alpha}{\Rightarrow} q$.*

The behavioural equivalence, denoted by $\sim^{NC}$, is an obvious extension of the interleaving one: bisimilar processes must perform the same concurrent paths with fresh bound names and consistent link's labels. However, it has the additional property of being a congruence.

Theorem 3. *$\sim^{NC}$ is a congruence with respect to all NCPi operators.*

Proof (Hint). This is proved by considering each possible elementary context and defining a suitable bisimulation closed under that context. The difficult case is the input prefix, since a renaming, possibly not injective, is involved. The key idea is that renaming a process may allow to apply more (COM), (SRV-IN), (SRV-OUT) or (SRV-SRV) rules, but the paths these rules concatenate are already observable in the original process, so the new transitions only depend on the original process' ones. In the appendix an outline of the proof of $\sim^{NC}$ being closed under all renamings can be found. □

The rules in Fig.5 generate a concurrent version of the π-calculus transition system, via the encoding of Proposition 2.

Corollary 1. *The bisimilarity on the concurrent π-calculus transition system is a congruence.*

This result is analogous to that in [14] but, as already mentioned, there the synchronization mechanism is not faithful to the π-calculus: in [14] the synchronization channel is observed unless restricted, for instance $\overline{a} \mid a \xrightarrow{\tau_a} \mathbf{0}$, while for our calculus $\overline{a} \mid a \overset{\bullet;\bullet}{\Longrightarrow} \mathbf{0}$, which corresponds to τ.

7 Conclusions

In this paper we presented NCPi, an extension of π-calculus with an explicit notion of network. To achieve this the syntax is enriched with named connectors. From a semantic point of view, an observation is a snapshot of the traffic on the network, represented as the paths concurrently covered by the data. The semantics' concurrent nature is the key feature which allows bisimilarity to be a congruence.

Related work. The works most closely related to ours are [8] and [10] where network-aware extensions of $D\pi$ [9] and KLAIM[11] are presented, called respectively $D\pi_F$ and TKLAIM. KLAIM is quite far from the synchronous π-calculus, because it models a distributed tuple-space modifiable through asynchronous primitives, but an encoding to the asynchronous π-calculus exists [16]. Both $D\pi_F$ and TKLAIM are *located* process calculi, which means that processes are deployed in *locations*, modelling physical network nodes. In NCPi, instead, processes access the network through sites, possibly more than one for each process, rather than being inside of it. However, locations can be easily introduced in NCPi by a typing mechanism which limits the number of subject names in processes. The network representations are quite different: 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 section 2; this example, in our opinion, is not easily implementable in TKLAIM and $D\pi_F$. Finally, our calculus is more programmable: processes explicitely activate transportation services over connections via the link prefix, while in the cited calculi the network is always available.

We can also cite [17–19] as examples of calculi where resources carry some extra information: they explicitely associate costs with π-calculus channels through a type system. In our case, links could also be typed in order to model services with different features, e.g. performance, costs and access rights.

Research directions. Our calculus only captures point-to-point communication, but a network could be used for more complex forms of interaction, e.g. multicast. One possible development direction might be allowing different mechanisms of message exchanging. Moreover, one can think of complex conditions on resources regulating the coexistence of paths, e.g. restrictions on bandwidth or costs. There is also some room for asynchronous variations, for instance each hop of a routing path could be performed in different transitions. This would capture the behaviour of SDNs in a more faithful way.

Network-awareness is only one form of resource-awareness, which is essential to adequately model new computational paradigms such as cloud computing. Future work includes also the development of an algebraic/coalgebraic categorical model of resource-aware nominal calculi. In particular, the approach based

on presheaf models has been successfuly applied to the π-calculus [20], the fusion calculus [21] and the explicit fusion calculus [22]. This approach is especially effective for nominal calculi, because it allows to model resources as a separate index category, so to decouple the structure of resources from the syntax and semantics of processes using them. This permits to capture many alternatives with minimal changes. Moreover, coalgebras over a broad class of presheaves can be implemented as HD-automata [23, 24], more concrete operational models that allow for name deallocation and hence are suitable for verification purposes. In our case, the resources of a process are its free sites and links, which can be represented as a finite graph. Functors on the category of resources could allow to create new sites and new links, and to increase their capabilities, similarly to what happens with functor δ in the presheaf semantics of the π-calculus.

References

1. : Openflow foundation website. http://www.openflow.org/
2. Campbell, A.T., Katzela, I.: Open signaling for atm, internet and mobile networks (opensig'98) (1999)
3. Tennenhouse, D.L., Wetherall, D.J.: Towards an active network architecture. Computer Communication Review **26** (1996) 5–18
4. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G.M., Peterson, L.L., Rexford, J., Shenker, S., Turner, J.S.: Openflow: enabling innovation in campus networks. Computer Communication Review **38**(2) (2008) 69–74
5. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100**(1) (1992) 1–40
6. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, ii. Inf. Comput. **100**(1) (1992) 41–77
7. Milner, R.: A Calculus of Communicating Systems. Volume 92 of Lecture Notes in Computer Science. Springer (1980)
8. Francalanza, A., Hennessy, M.: A theory of system behaviour in the presence of node and link failure. Information and Computation **206**(6) (2008) 711 – 759
9. Hennessy, M., Riely, J.: Resource access control in systems of mobile agents. Inf. Comput. **173**(1) (2002) 82–120
10. Nicola, R.D., Gorla, D., Pugliese, R.: Basic observables for a calculus for global computing. Information and Computation **205**(10) (2007) 1491 – 1525
11. Nicola, R.D., Ferrari, G.L., Pugliese, R.: Klaim: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. **24**(5) (1998) 315–330
12. Corradini, A., Montanari, U., Rossi, F.: An abstract machine for concurrent modular systems: Charm. Theor. Comput. Sci. **122**(1&2) (1994) 165–200
13. Lanese, I.: Synchronization strategies for global computing models. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy (2006)
14. Lanese, I.: Concurrent and located synchronizations in i-calculus. In van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plasil, F., eds.: SOFSEM (1). Volume 4362 of Lecture Notes in Computer Science., Springer (2007) 388–399
15. Y.Rekhter: A border gateway protocol 4 (bgp-4). http://www.ietf.org/rfc/rfc1771.txt (March 1995)
16. De Nicola, R., Gorla, D., Pugliese, R.: On the expressive power of klaim-based calculi. Theor. Comput. Sci. **356**(3) (2006) 387–421

17. Hennessy, M.: A calculus for costed computations. Logical Methods in Computer Science **7**(1) (2011)
18. Hennessy, M., Gaur, M.: Counting the cost in the picalculus (extended abstract). Electr. Notes Theor. Comput. Sci. **229**(3) (2009) 117–129
19. De Vries, E., Francalanza, A., Hennessy, M.: Reasoning about explicit resource management (extended abstract). In: PLACES: Programming Language Approaches to Concurrency and Communication-cEntric Software, ETAPS (April 2011) 15–21 http://places11.di.fc.ul.pt/.
20. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: LICS. (2001) 93–104
21. Miculan, M.: A categorical model of the fusion calculus. Electr. Notes Theor. Comput. Sci. **218** (2008) 275–293
22. Bonchi, F., Buscemi, M.G., Ciancia, V., Gadducci, F.: A category of explicit fusions. In: Concurrency, Graphs and Models. (2008) 544–562
23. Ciancia, V., Montanari, U.: Symmetries, local names and dynamic (de)-allocation of names. Information and Computation **208**(12) (2010) 1349 – 1367
24. Ciancia, V., Kurz, A., Montanari, U.: Families of symmetries as efficient models of resource binding. Electr. Notes Theor. Comput. Sci. **264**(2) (2010) 63–81

Appendix

Outline of the proof of $\sim^{NC}$ being closed under all renamings. We prove that the relation $\mathcal{R}$ defined by applying all renamings to all pairs of bisimilar processes is a bisimulation.

Consider two processes $(p, q) \in \mathcal{R}$. For the sake of simplicity we treat the case of σ being an elementary renaming $[r'/r]$, but the general case is an obvious extension. We assume $r, r' \notin \mathrm{bn}(p) \cup \mathrm{bn}(q)$ (this can always obtained by α-conversion).

Suppose $p\sigma \xrightarrow{\Lambda} p'$ and consider a proof P for this transition: P contains some rules "triggered" by σ, i.e. (COM), (SRV-IN), (SRV-OUT) and (SRV-SRV) that further concatenate paths which had different interaction and/or destination sites as performed by p, but then these sites are identified by σ. These rules can be moved at the end of the proof by making them "jump over" their following rules. This might require modifying the rules, e.g. if we have a (COM) followed by a (RES) that restricts the objects of the two paths (COM) concatenates, and we want to invert the order of these rules, first we have to turn the (RES) into an (OPEN), because now the two paths are separate in its premise and their objects are observable, and then put (COM) closing the scope of these objects.

It can be proved that jumping is allowed, by considering each pair of consecutive rules where the first one is a concatenation rule. The critical case is when the second rule adds a new restriction, because delaying the concatenation of two paths α_1 and α_2 causes more names to be observed (namely their interaction and destination sites) and this in principle might violate the rule's side conditions. However, the violation cannot happen, because these names are equal to r', which we assumed not bound in p.

In the end we clearly get a transition of $p\sigma$ with the same label Λ, but its continuation may be different from p'. In fact, moving some (COM) to the end

of the proof may bring some restrictions at the top level in p'. However such a continuation would be bisimilar to p', since the former could be obtained from the latter just by applying the axiom of scope extrusion, which definitely holds for our bisimilarity.

The new proof has two parts P' and P'', where P'' contains only the concatenation steps we moved. Notice that the steps in P' are all and only those applicable not only to $p\sigma$ but also to p. Being p and q bisimilar, also q can perform a corresponding transition, with the two continuations p'' and q'' being in relation $\mathcal{R}$. Finally, if we apply σ to processes and paths of the latter transition, we can apply the steps in P'' and thus we can simulate the transition from $p\sigma$ to p' with a transition from $q\sigma$ to q'.

Now, observe that if we apply any sequence of concatenation rules to a transition, we get a transition whose continuation is the original one, possibly with some additional restrictions and renamings, because the sequence may contain some (COM) rules closing the scope of some names.

Since p' and q' are obtained from p'' and q'' by applying σ and then adding the same renamings and restrictions, and since it is easy to prove that restriction preserves bisimilarity, we conclude that also p' and q' are in relation $\mathcal{R}$. $\square$