A Human-Inspired End-to-End Principle for the Internet

ABSTRACT

The vision of the Knowledge Plane, followed within subsequent works, recognizes the need for a different set of abstractions being used for distributed system design: one that allows systems to scale while adapting to evolving conditions at the same time. In this paper, we present our contribution to this area by introducing our human-inspired system design approach and by defining a functional model that is based on notions derived from problem solving. In our view, this shifts the focus of the communication system onto disseminating information pertaining to realizing the solution to any given problem. With this, we provide a common nucleus upon which we can assemble large-scale distributed systems. This assembly involves the reconciliation of possibly conflicting dissemination strategies that underlie each solution. We outline this functional model and present an initial proof-of-concept to show its applicability.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *distributed networks, network communications.*

General Terms

System design, network architecture.

Keywords

End-to-end, information-centric, knowledge plane.

1. INTRODUCTION

The Internet has been vastly successful in providing ubiquitous means for communication between people and between machines as well as for information retrieval. Such success is built on a simple paradigm, namely that of facilitating the exchange of messages between addressable endpoints for the purpose of resource sharing and information retrieval.

However, the initial simplicity has become ever more complicated as many functions are now involved in the process of communicating between applications. They range from address space management and authentication over various policing functions to attempts of quality assurance. This increased complexity negatively affects the initial goal of simple and transparent communication between any given endpoints. It is perfect when it works – it is frustrating for end users and service personnel alike when it does not.

Hence, the question arises as to *what are the right abstractions for designing a large-scale distributed system (such as the Internet) that are simple yet adaptive to the evolving needs and goals of participating actors*?

One of the most prominent works addressing issues surrounding this question is the Knowledge Plane [1], which recognizes this frustration and envisions *"…to build a fundamentally different sort of network that can assemble itself given high level instructions, reassemble itself as requirements change, automatically discover when something goes wrong, and automatically fix a detected problem or explain why it cannot do so.*" [1]

In order to implement such vision, one needs to reconcile two main requirements. Firstly, a solution must be able to reason over its proper functioning based on a defined operational goal. The result of such reasoning can be the (partial) reconfiguration or at least a meaningful notification of end users and operators alike when detecting malfunctioning. This is seen as the *cognitive* part of this vision, requiring solutions for online as well as offline reasoning over a set of useful information within a context of a defined goal. All of this has to be implemented in the presence of the second requirement, namely *scaling* to the size of the current (and any future) Internet. This requires solutions that can perform at ever increasing line-speeds. It seemingly creates a dichotomy of opposing requirements that are difficult to fulfil in any realization of this vision.

In our attempt to provide a solution to this dichotomy, we find inspiration in efforts on *meta-reasoning architectures*, and more specifically in the architecture defined by Chappell and Sloman [2]. In this work, the authors attempt to capture the meta-reasoning and deliberative processes in human cognition. The authors devise a functional architecture that integrates the *reactive processes* that we humans so quickly and subconsciously execute in everyday activities with the more conscious processes of *goal-oriented reasoning*. Most operations of this system are executed as 'fast path' operations, while slower meta-reasoning and deliberation only occurs when an attention filter threshold is exceeded, i.e., when attention by the more complex reasoning processes is deemed necessary.

Let us now apply the Chappell/Sloman model to distributed systems by focusing on solving *problems* that arise throughout such systems. Paraphrasing [3], problems involve *"a collection of information that"* an implementation *"can use to decide what to do"*, which is to implement a problem solution. The basic information required includes states, goals, and actions. Problems have a nature of spawning sub-problems or relate to other problems. Hence, they can be *concatenating* or *inclusive*.

When solving problems through distributed systems, *computation* and *storage* resources are appropriately utilized. We facilitate such resource utilization through the exchange of any relevant *information*. This makes information the core of our approach[1]. For that purpose, information is explicitly identified. This is extended by the notion of a *scope* that defines the *set of information* being

[1] This is a significant departure from the IP paradigm in which the location of endpoints is at the centre, with opaque bits being transmitted between these endpoints.

disseminated for the problem solution at hand as well as the *realization of the basic dissemination functionality* that is specific for the problem and its information.

Such focus on information and scopes forms the core of the work in this paper, with its main contribution to fully define an information-centric functional model that serves as a nucleus for large-scale distributed systems within a problem solving framework that is inspired by the Chappell/Sloman meta-reasoning architecture (see Figure 1). With that, we formulate a system design approach that can form the basic blueprint for a variety of architectures.

We also formulate an end-to-end design principle that bases the assembly of complex solutions on the reconciliation of the various dissemination strategies that underlie each individual solution. We envision such reconciliation being codified at design time in the form of standards, specifications, and running code. We furthermore extend the principle towards an assembly of functions at system runtime. This allows for combining our functional approach with proposals on dedicated control planes as well as programmable network nodes [4][5][6]. Such runtime extension reconciles the co-existence of (slower) cognition and fast operation within a system that can optimize its operation at runtime. For this to happen, we envision an increasingly formal specification of the dissemination strategies, such as through using Semantic Web techniques.

Figure 1 visualizes our framework. Each circle indicates a solution to a problem, implementing our functional model. The connecting elements between reactive and deliberative processes are *attention filters* that trigger when given thresholds are exceeded.

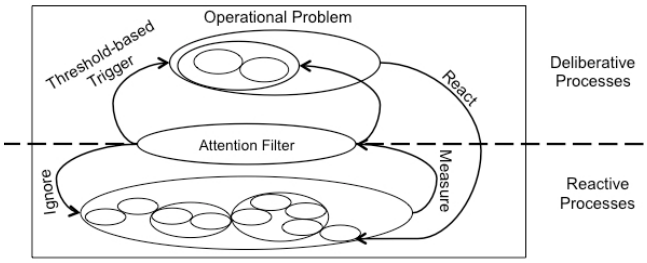

Figure 1: Problem Solving Framework

Comprehensively understanding and evaluating the applicability of our framework is a challenging task. This paper can only be the start of creating such understanding. We take a rather mundane approach by focusing on the presentation of the framework itself and the evaluation of a possible artifact that we can create with it. For this, we extend on our motivation in Section 2, before outlining the functional model as well as the assembly of complex functions in an end-to-end manner in Section 3 as the basic element of our framework. A proof-of-concept of this basic element is presented in Section 4 with an initial evaluation of its performance. In Section 5, we then discuss how to utilize this nucleus to achieve the creation of deliberative processes, outlining the various issues that need future work. In Section 6, we then turn to the aspect of enabling various architectures beyond our given example in the paper. With this, we set out a research agenda that could lead to our approach being the common basis for a variety of architectures.

2. INPUT INTO OUR WORK

There are many efforts that provide input into our work. Most notably, the knowledge plane in [1] outlines the vision to utilize cognitive techniques rather than traditional algorithmic approaches for the assembly of functions in a distributed network. The cognitive approach taken by the authors has sparked a number of research efforts. Some of these works, such as in [4][5][7][8], are anchored in a particular layered model that introduces a control plane in addition to an endpoint-centric data plane that is IP today.

Others utilize a range of techniques from the Semantic Web [9][10] area, addressing the notion of *cognition* by using ontological concepts. While some of these efforts are directed towards the application layer, the work in [11], among others, proposes a new architectural approach to include reasoning and perception into the lower layers of distributed systems, albeit with a similar 'control plane' approach that is suggested in many other works.

Another branch of research efforts, such as those of [12][13], introduce the notion of *autonomy*. In such systems, individual actors would implement functions autonomously. It is the network architecture that provides means to reason over conflicts that arise through such autonomous actions, and mediate possible outcomes. For this, most autonomous network approaches borrow concepts from biology and other disciplines. The authors in [14] outline a variety of challenges. Many of these are related to the problems of information modelling and reconciliation of arising conflicts. We later see that these challenges are to be addressed in our approach, too.

Many of the above mentioned efforts take the view of extending networks with a cognitive ability, requiring a sense of *programmability* in the network elements. The increasing work on programmable routers directly targets the enablement of such programmability. Today, this is mostly driven by the OpenFlow efforts [6] that aim at providing a common basis for programmable router platforms. These efforts are extended by high-level declarative approaches to programming such elements, as in [15] or [16], as well configuring them [17].

A very different input into our work comes through Sollins in [18]. In this work, the author argues for a notion of regions. These define a sense of *differentiation* that can relate to trust issues, locality of problems, or a combination of technologies being used. Sollins asserts that designing for differentiation would allow achieving scalability of the solutions being provided. This line of thinking is close to our concept of scoping problems and their solutions.

Arguments such as presented in [19] question the existence of a single Internet architecture that forms a common basis for an operational and large-scale communication system.

Instead, the existence of a pluralistic environment with many architectural foundations is envisioned, brought about by virtualizing a common substrate that allows for these various foundations to unfold. This brings us to another area of input into our work, namely that of describing large-scale distributed systems as an *assembly of functions* in a possibly varying set of constellations that one could call an 'architecture'. Outlined as a scientific approach to network design, Day argues in [20] that networking is nothing else than a form of inter-process communication that leads to a pattern of functions being executed with differing scope and extent. Touch et al., in [21], take this notion further by outlining a functional model for a distributed networked system that is assembled recursively rather than layered in the traditional sense. Other works, such as [22][23], translate this approach onto information-centric network approaches.

It is the latter work that drives the formulation of a functional model at the heart of our work, while the work on cognition and the vision of the knowledge plane provides the context of assembling these functions into a large-scale, end-to-end networked system.

3. A COMMON NUCLEUS FOR DISTRIBUTED SYSTEMS

At the heart of our problem solving framework in Figure 1 is the assembly of problem solutions towards a large-scale distributed system. In the following, we outline the common nucleus that enables such assembly, starting from invariants in Section 3.1 over the functional model for each individual problem solution in Section 3.2 towards an end-to-end principle for assembling individual solutions into a larger system in Section 3.3.

3.1 From Invariants…

We formulate five invariants as the basis for our functional model. The first one is that of *referencing* being the means of identifying *information items* as the main entities. An information item is generally any form of data that is relevant within a particularly given problem solution. It can represent a principal of a transaction, a policy rule acting on another piece of data or a pointer to some imaging data. Each information item is identified using a (statistically unique) flat label identifier. These identifiers are self-generated and the associated semantic of the information is only known to the problem solutions generating said identifier. As an example, a video publisher might generate an identifier through hashing a human-readable name into a suitable identifier.

The second invariant is that of *scoping*. Scopes define the set of information that is being disseminated for the implementation of a particular problem. With that, each scope represents the boundaries of a defined dissemination strategy for the information it contains. This achieves scoping on *functional* and *information* level. Each information item is being placed in at least one scope. With scopes being a set of information, they are treated as information items themselves. They can therefore be placed in a scope itself, allowing for building complex graphs of information (see Figure 2). Each item is identified by a specific ID and contains some data (and possibly metadata). For illustration, we denote individual pieces of information with *Rendezvous identifiers* (*RIds*), each of which are (statistically) unique within the scope to which it is assigned. Scopes are identified through a *Scope identifier* (*SId*), which is again (statistically) unique within the scope in which it is placed. The scope's data contains the identifiers of all underlying information structures, including its sub-scopes[2]. As shown in Figure 2, different pieces of information with the same RId (RId_3) can be published under different scopes. Moreover, the same piece of information (e.g., RId_4) may be published under two different sub-scopes. Equally, a scope (e.g., SId_3) may reside under two different super-scopes.

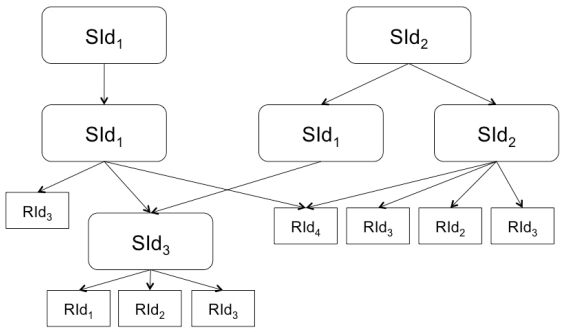

Figure 2: Information Scoping

The third invariant is the *service model* as that of publishing of and subscribing to individual information items within a scope. Traditional request-response service models can be implemented by, for instance, including identifiers for response information into the original request publication with the client subscribing to the response identifiers and the server publishing to it.

The fourth invariant is that of *separating the functions* for disseminating information within a given scope into three distinct ones. The first function, *rendezvous*, concerns the matching of publishers' availability of information and subscribers' interest in it. This matching results in some form of location information for the set of publishers and subscribers that has been matched. This information is used

[2] It is important to understand that the invariants of scoping and referencing do not imply particular identifier structures. For instance, flat label namespaces can be utilized as well as hierarchical naming structures. The particular design considerations are outside the scope of discussion for the general functional model. The requirement of statistical uniqueness is introduced to enable self-created identifiers, avoiding the necessity for system-wide governance being imposed on the most basic form of identification. This requires methods of conflict resolution per scope that need to be developed. We could foresee, however, the definition of a governed information space within each dissemination strategy, e.g., through the assignment of labels to specific subscribers.

by the second function, *topology management and formation*, to determine a suitable topology for the transfer of the respective information, this transfer being executed by the third function, *forwarding*. The exact nature and implementation of these functions is defined by the dissemination strategy of the scope for which these functions are realized. Hence, together with the scoping invariant, this establishes a functional scoping through which these distinct functions can be optimized towards the particular problem that is being solved.

The fifth and last invariant is that of the *dissemination strategy* that underlies each scope. Such strategies define the methods being used for implementation, such as realizing the functions of rendezvous, topology formation and forwarding in dedicated (centralized or distributed) components, utilizing particular data representation formats and alike. By default, the defined policies and strategies of a scope apply to all information items within the given scope, including all sub-scopes[3]. The dissemination strategy is usually codified through the processes of engineering the solutions for our problems at the *design time* of the system. These codifications include the specification of the functionality being implemented, including that of assembling larger problem solutions through the assembly of smaller ones. Section 3.4 outlines examples for such assembly of problem solutions.

It is important to note that the information referencing invariant gives the ability to 'relate' problem solutions with each other through 'linking' in the form of *metadata*. For instance, in order to implement a domain-local dissemination within one scope, said domain might need to maintain information that is global (e.g., for the very routing of the application-level information). This might require its own problem solutions, such as for maintaining links and router information, that relate to the problem of our original scope. Such relation can be done via linking, i.e., information items within the original scope might contain identifiers from other scopes (including the scope identifiers themselves). Hence, problems (and their solutions) usually exist in parallel and they might relate to each other but they are not directly built onto each other (which would result in sub-scoping).

3.2 …Over a Functional Model…

The invariants can be put together in a functional model, as outlined in Figure 3. It shows the functional and information scoping that is provided through combining the first, second and fourth invariant and it exposes the service model defined in our third invariant. What defines a coherent region of the functional model is the dissemination strategy, defined in our fifth invariant. The element of sub-scoping enables recursively assembling the functional model towards complex distributed systems, as indicated in the nested structure in Figure 3. It is this functional model that provides the nucleus for a single problem solution that is being implemented.

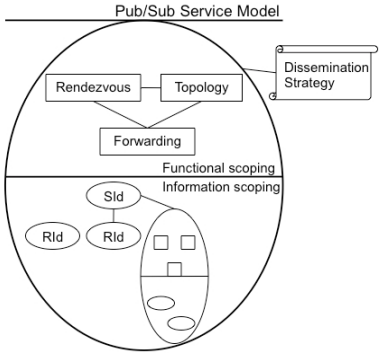

Figure 3: Functional Model

3.3 …To An E2E Principle

In order to implement end-to-end distributed systems as an assembly of individual problem solutions, we formulate the following end-to-end principle as the foundation. In such assembly, the notion of a dissemination strategy is crucial.

> *A given problem within a distributed system can be implemented through an assembly of sub-problem solutions, whose individual dissemination strategies are not in conflict with the ones set out by the problem in question.*

It is important to understand that this design principle states the realization of distributed applications within our architectural framework as a process of reconciling various dissemination strategies. At design time, such reconciliation process is represented through methods of requirements engineering, functional specification, and standardization. Hence, the reconciliation is the result of the 'satisficing'[4] process that these methods represent. In Section 5, we investigate how to extend this process towards a dynamic reconciliation process, enabling the implementation of deliberative processes within our problem solving framework of Figure 1.

Although not explicitly addressed, the formulated functional model and the described assembly towards a complex distributed system shows that we do not assume a strong layering principle similar to that of the current Internet. Instead, our functional model revolves around the loose assembly of regionalized information dissemination with tasks being concatenated or being implemented as inclusive subtasks. As long as the dissemination strategies of the various subtasks are aligned, their assembly towards increasingly complex distributed problem solutions is in compliance with our end-to-end principle. With that, we take a different approach towards the issue of 'cross-layer violations' in traditional IP architectures. In our framework, as long as the various operations do not violate other strategies, they are entirely legitimate.

[3] Dissemination strategies, however, can be overridden according to the super-scope's policy.

[4] A hybrid between the words *satisfying* and *sufficing*, see http://en.wikipedia.org/wiki/Satisficing

It is worth stressing that such loose assembly is similar to today's Internet in some respect. Tens of thousands of individual network domains assemble their communication resources towards a unified packet transport service that is IP today, while optimizing their internal operations towards the high-level dissemination strategy of global connectivity. In that sense, it is not surprising that the IP design adheres to our design principle, too[5].

3.4 Assembling Problem Solutions

Our functional model together with our E2E design principle now gives us the ability to assemble problem solutions for larger systems.

For a node-local scenario, we merely assume a single rendezvous function towards the application that maintains and manages the information space that is only locally accessible within that node. For a link-local dissemination, we envision the information space to be managed by a rendezvous function running in a selected node on this link with the information being only "visible" to network nodes physically connected to that node.

Extending to a more complex domain-local dissemination strategy, the management of the domain topology is likely to be solved separately from the actual delivery of application-level information. For this, we envision one or more dedicated topology management nodes to subscribe to specific (administrative) scopes in order to receive topological information from dedicated forwarding nodes. End nodes participating in the domain-local strategy could subscribe to scopes created by these topology managers to request appropriate link identifiers (or entire graphs of such identifiers) from a set of end nodes acting as publishers for application information to another set of end nodes acting as subscribers to this information.

It is this concatenation of problem solutions that is generally enabled by our framework. Section 4.4 outlines the particular dissemination strategies that we have implemented to date in our proof-of-concept.

4. PROOF OF CONCEPT

The following section provides an initial implementation of our common nucleus. With this, we intend to demonstrate that it is implementable with a reasonable performance. Given the information-centric nature of our framework, our implementation focuses on realizing the information management and interfaces for the functional model. We do not optimize memory management and therefore high performance of our network nodes, leaving this for a thorough software engineering exercise at a later stage.

As a basis for our implementation, we choose the Click software router [24], which provides reasonable speed for a proof-of-concept demonstrator while enabling future extensions rather easily. Section 4.3 more specifically outlines how we realize our implementation on this node architecture. But first, we present the realization of the node-internal information management and the provided service interface towards applications.

[5] However, there are numerous conflicts through internalized operations, such as address re-writing and others, that come into conflict with the overall goal of global connectivity; an observation that is pointed out in our introduction.

4.1 Realizing the Information Management

Implementing our first two invariants of Section 3.1, the information structure supported by our prototype is that of a directed acyclic graph. Leaf nodes in the graph represent individual pieces of information, while inner nodes represent scopes.

Figure 4 presents an example of a potential information structure in which two root scopes, SId_1 and SId_2, exist. All scope identifiers (SIds) assigned to root nodes are statistically unique. All "children" of a scope, i.e., all its subscopes and information items, are assigned with locally (statistically) unique scope and rendezvous IDs, respectively. As observed in Figure 4, scopes and information items that are not "children" of the same scope may carry the same IDs. Within our implementation, each node in the graph is identified with its full path starting from a root scope. For instance, the rightmost item is identified as $/SId_2/SId_2/RId_3$.

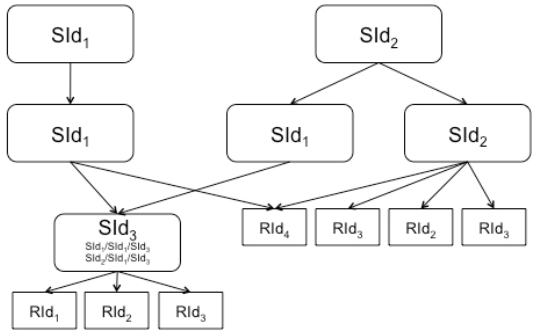

Figure 4: Information Structuring

The information structure is maintained according to the dissemination strategy that is assigned to each topmost scope. For instance, in a node-local strategy, the rendezvous component of the local node (see Section 4.3 for the system components of our implementation) maintains the information structure. In a domain-local strategy, it is likely that a dedicated network node (or a set of such nodes) is responsible for maintaining the structure. Section 4.4 outlines the currently implemented dissemination strategies of our prototype.

The dissemination strategy itself is stored as metadata within the information structure. This will help reconciling strategies when they are changed throughout the existence of the information structure. Other metadata information is group membership information, possible access control information as well as possible application-specific policy that apply to a branch of the information structure.

4.2 The Service Model

We now address the third invariant of Section 3.1 by outlining the implemented service model of our prototype.

4.2.1 Publishing and Republishing Scopes

The *create_scope(String ID, String prefixID, Strategy st)* function creates a new scope in the information structure. Both the *ID* and the *prefixID* may contain a variable number of identifiers (SId or/and RId). If the prefixID is null and the ID represents a single ID fragment, then the system will try to create a root scope in the information structure. Depending on the dissemination strategy *st*, the scope's *Rendezvous Point (RP)* may be created locally or in a remote host (e.g., in a dedicated RV node).

Part of the operation is a mechanism to decide whether the operation should be accepted or not. Currently, we do not implement any particular mechanism but envision access control functions to be integrated at a later stage. Consequently, every node in the system is currently able to create and delete any scope.

If the method is passed an existing prefixID and a single-fragment ID, the system will create a new inner scope with ID */prefixID/ID*. Finally, if both the *ID* and the *prefixID* contain multiple IDs, the system will republish a scope under an existing scope.

Publishing and re-publishing scopes triggers a notification towards existing subscribers. The hierarchical structure of the information allows to subscribe to a scope and to receive notifications whenever a new scope is created or a new piece of information is published (or advertised) underneath that scope. In our prototype, the RV component notifies all potential subscribers. In the case of republishing scopes, a subscriber receives the path ID that contains the scope to which it was previously subscribed.

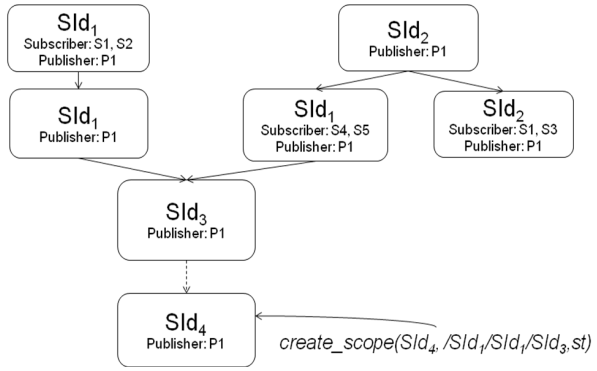

Figure 5: Publishing a Scope

In the example of Figure 5, a new scope is published under scope */SId_1/SId_1/SId_3/* by publisher *P1*. The list of subscribers that was previously subscribed to specific scopes is also depicted in Figure 5. In this example, the system notifies subscribers *S1*, *S2*, *S4* and *S5* about the new scope. Subscribers *S1* and *S2* are notified about a scope with ID /SId_1/SId_1/SId_3/SId_4, while subscribers *S4* and *S5* receive a notification about a new scope with ID /SId_2/SId_1/SId_3/SId_4.

4.2.2 Advertising Pieces of Information

The *advertise_info(String ID, String prefixID, Strategy st)* function creates a new piece of information in the information structure. If the *ID* contains a single ID fragment, then a new information item is created under the scope identified by the *prefixID* (subject to the existence of that scope). If the *ID* contains multiple ID fragments, the existing piece of information is also published under the scope identified by the *prefixID*. Note that this method does not accept actual data from the application. The actual transfer of data happens after the RV component notified the publisher of matching subscribers. With that, the end-user applications are responsible for storing all information items until one or more subscribers for these items appear.

According to the dissemination strategy *st*, the piece of information will be published under a scope in the respective RV point, subject to the compliance of *st* with the strategy of the "father" scope.

4.2.3 Unpublishing Scopes and Information

The *unpublish_info(String ID, Strategy st)* function triggers the system to delete all references to a specific scope or piece of information that is uniquely identified by the *ID,* subject to the compliance of *st* with the scope's strategy. More specifically, if the *ID* identifies a scope, the system checks if the scope is published under more than one scopes (i.e., if it has more than one "father" scope). If this is the case, then the RV system simply deletes the reference from the "father" scope identified in the *ID*. Otherwise, all descendant scopes and information items are checked. If there are no remaining subscribers or publishers, all descendants are deleted along with the scope identified by the *ID*. In any other case, the RV function only deletes the scopes and the previously advertised information items that fulfil the abovementioned criteria. For all deleted scopes, their references are also erased from their parent scopes.

4.2.4 Subscribing to Scopes and Information

The *subscribe_to_scope(String ID, bool sub, Strategy st)* function subscribes the calling subscriber to the scope identified by the *ID*.

As shown in Figure 6, subscriber *S1* requests to subscribe to scope /SID_1/SID_1. Assuming that *S1* is authorized to subscribe to this scope, then the RV function implicitly subscribes *S1* (*S1* is shown in grey color) to all descendants of it, if requested through setting the *sub* parameter to *true* (as it is in Figure 6). With that, *S1* is notified regarding the scopes /SId_1/SId_1/SId_3 and /SId_1/SId_1/SId_3/SId_4. Moreover, *P1* is notified about the existence of subscribers for the information items advertised under the latter scope.

A subscriber can subscribe to a piece of information by calling the *subscribe_to_info(String ID, Strategy st)* function. With that, the system verifies if there are any publishers for that item. If so, the publishers are notified, as described above. Otherwise, the interest is registered.

Both calls assume that either the scope or the information item with the given *ID* exists or that at least their parent scope exists. If only the parent scope exists, the new scope or information item is created and the interest of the subscriber is registered. In any other case the call is

rejected. Moreover, both calls succeed only if strategy *st* is compliant with the dissemination strategy of the scope or the "father" scope of the information under subscription.

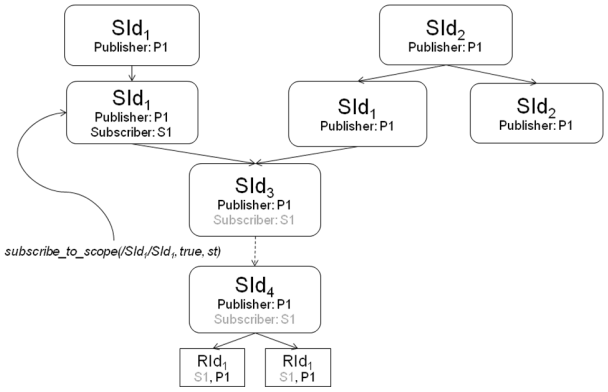

Figure 6: Subscribing to a Scope

4.2.5 Unsubscribing from Scopes and Information

Unsubscribing from a scope or a specific piece of information is realized by calling the *unsubscribe(String ID, Strategy st)* function. The system verifies that strategy *st* is compliant with the scope's dissemination strategy and that no other publishers or subscribers exist within the unsubscribed scope or any subscope. If so, the scope is removed from the information structure. The same check is performed when unsubscribing from an information item.

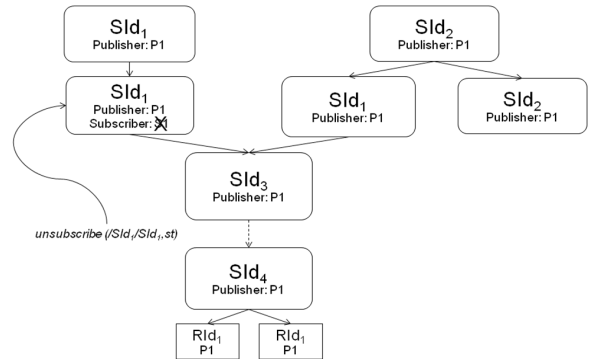

Figure 7: Unsubscribing from a Scope

Figure 7 presents the information structure after the scenario presented in the previous subsection. Although *S1* is only added to scope /SID_1/SID_1, whenever new scopes are created or new pieces of information are advertised under that scope the RV function will check all ancestors in the graph, therefore including *S1* in the list of subscribers. When subscriber *S1* unsubscribes from that scope, it is deleted from the list of subscribers for that scope. The RV function does not delete the entry for that item, however, since active publishers still exist.

4.2.6 Publishing Information to Subscribers

The final act of a publish/subscribe service model is that of publishing the actual data corresponding to a specific piece of information to one or more subscribers by calling the *publish_info(String ID, Strategy st, char *data)*.

Note that the behaviour of this call depends on the dissemination strategy of the scope under which the information item is published. Usually, the data is sent following a notification of interest by the RV component. A dissemination strategy could, however, foresee the publication of data without prior explicit notification. This could, for instance, be the case in dissemination strategies that assume a rather stable set of subscribers or a single subscriber (e.g., in a voice or video call scenario).

4.2.7 Setting Dissemination Strategies

As outlined in Section 4.1, individual dissemination strategies can be assigned to each of these information structures. This is currently done via the *create_scope()* function. However, we foresee a system interface similar to the *setsockopt()* function of the socket API that would allow for changing dissemination strategies after the scope creation. It is the task of the node protocol stack to consolidate the dissemination strategies within the information structure in order to avoid conflicting strategies (e.g., by inheriting any change in dissemination strategy for a given scope in all included sub-scopes). A node's protocol stack might have a default dissemination strategy that depends on the expected usage. For instance, there could exist a default wireless strategy for a mobile device. The support for this function is left for future work.

4.3 Node Implementation

Our current implementation is based on the Click Router architecture [24]. Figure 8 shows the architecture of a single network node in our system.

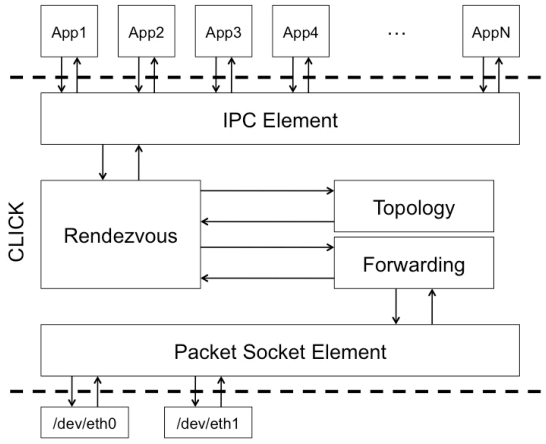

Figure 8: Node Implementation Architecture

Each node contains the Click elements depicted in Figure 8. The *IPC Element* implements an asynchronous inter-process mechanism so that user-space applications can issue publish/subscribe requests to the Click software. Via this mechanism, applications may communicate with each other, using the service model described in Section 4.2. Currently, the IPC element implements a TCP socket server, although other techniques, like Unix sockets or shared memory, could be incorporated in the future. Click components are single-threaded. Therefore, the IPC element has to run in the single thread provided by the

Click execution environment. A selection mechanism (based on the *select()* system call) is implemented for the descriptors that correspond to duplex channels between the Click platform and the applications.

The lower part in Figure 8 consists of the *Packet Socket Element,* which is responsible for sending and receiving link-layer frames. The current implementation is based on existing Click elements, creating Ethernet frames and forwarding them to the appropriate network interface. The Packet Socket Element is based on the PF_SOCKET implementation that is supported by many operating systems. In addition, we provide the ability to utilize raw IP data packets as an alternative packet forwarding mechanism. This provides the ability to test our prototype in Internet-wide scenarios.

The functions for *rendezvous* (RV), *topology* management (TM) and *forwarding* are the core elements of our network node, addressing the fourth invariant of our functional model. Their implementation depends on the dissemination strategy that is supported by each network node. Section 4.4 outlines the current strategies of our prototype and therefore the differences in implementing these functions.

In general terms, the RV function implements the matching functionality for all scopes for which the particular network node is authorized to host the rendezvous function. The TM function runs in each network node. Its responsibility is to enrich the scope data structures (see Section 4.1) with appropriate forwarding information once a match between publisher(s) and subscriber(s) has been found. Our implementation currently uses the iGraph library [25] for computing shortest paths between network nodes. Finally, the forwarding function receives link-layer frames from the network (via the Packet Socket Element). Depending on the forwarding information that accompanies each frame, it forwards them to other network interfaces and/or to the local RV function, which in turn forwards the frames to interested applications.

Our node architecture is a minimal implementation that aims at realizing the core functional model of Section 3, with the chosen Click approach clearly favoring extensibility over performance. Possible extensions could be the caching of information items or functions for providing fragmentation. Combining the Click router basis with approaches to improve performance, such as through directly compiling Click elements onto NetFPGA hardware [26] or through partially replacing the Click elements with memory-optimized software modules, is a software engineering effort that needs to be undertaken to improve overall node performance.

4.4 Implemented Dissemination Strategies

The Click-based node architecture in Section 4.3 provides us with the basis for implementing a variety of dissemination strategies that tests our design framework. The following section provides a set of examples for such dissemination strategies. It is important, however, to understand that the given examples are not meant to be prescriptive regarding the presented technology and design choices – any final implementation can differ as long as the underlying functional model and the E2E design principle is preserved.

4.4.1 Dissemination within a Single Node

Within a single node, the RV function of our implementation merely disseminates information to the application entities that are connected via the IPC interface. The IPC socket here serves as the information that represents the dissemination strategy.

4.4.2 Dissemination over a Single Local Link

In this scenario, we assume physical nodes being directly connected in an Ethernet LAN. The RV function here is merely represented by the function of encapsulating and decapsulating any higher-level scoped information within the local link scope[6]. The topology formation function is implemented through the link management within the local Ethernet drivers, such that an existing link is interpreted as an established topology between both nodes. The link local Ethernet operations implement the forwarding function. Hence, the dissemination strategy is realized by these basic Ethernet operations with the machine-internal identification of the appropriate 'link' being the information associated to this dissemination; it represents a *forwarding identifier* for the items within this link local scope. Within our Click implementation, these basic Ethernet operations are provided through the Click framework and the Packet Socket Element function.

4.4.3 Dissemination within a Localized Graph

From our link-local model above, we can easily expand towards a localized graph of nodes. The approach in [27] presents a forwarding approach in which information is disseminated along a graph of nodes by virtue of a constant length identifier (called LIPSIN identifier in the following). This achieves multi-point delivery within said graph. For that, the approach denotes every link in the graph with its (locally unique) identifier and computes a Bloom filter based identifier of constant length. When now sending information from the publisher to a set of subscribers, the forwarding nodes along the graph compare the given identifier against their local outgoing link identifiers (this being implemented through an AND operation). In the case of a positive match, the incoming information is copied towards the positive outgoing link[7]. With this, multi-point

[6] Given the known location of both publisher and subscriber, an explicit rendezvous is not required. Although our prototype realizes an explicit RV function, the implementation for the link-local strategy merely forwards the data to the Packet Socket Element. That, in turn, directly sends the packet on the local link, accompanied by the forwarding identifier.

[7] The reader is referred to [27] for a more detailed description of the operation, including the handling of false positives, the establishment of virtual links and other techniques to make this approach a forwarding solution for metropolitan networks.

delivery within the given graph is accomplished, expanding our link-local dissemination to an entire graph.

As outlined in Section 3.4, we utilize a dedicated management scope for managing the domain topology, as shown in Figure 9 (denoted SId_5). This scope disseminates the necessary forwarding information (i.e., the LIPSIN identifiers for sets of publishers and subscribers within a domain). For this, our prototype implements a centralized network node for the overall topology management. It gathers the individual link identifiers of each forwarding node and computes appropriate LIPSIN identifiers between publishers and subscribers. The dissemination of this management information is orthogonal to the dissemination of the actual application-level information but utilizes the same functional model and dissemination strategies.

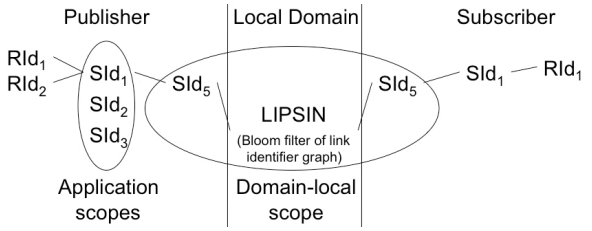

Figure 9: Domain-local dissemination

Furthermore, the RV function in a dedicated network node is used for matching publish and subscribe requests in domain-local scopes. The dissemination of information to this element is implemented via a dedicated LIPSIN identifier that describes the path to this node. After a positive publisher/subscriber match has been established for a given information item, the LIPSIN identifier encodes the graph from the publisher to the subscriber(s). It is this identifier that is used for disseminating the information at the local RV function within each node, being inserted into the data packet header at the Packet Socket Element level.

For now, we assume the LIPSIN identifiers towards the domain-local topology manager and RV function to be pre-configured, with a bootstrapping mechanism being implemented at a later stage.

4.4.4 Dissemination for Local Network Functions

The approaches for disseminating along local links and localized graphs can be extended towards the assembly of localized network problems. Examples for these are caching within a region of a network or the integration of hop-by-hop segmentation of information in mixed wired and wireless scenarios. For this, we envision the establishment of sub-scopes that implement the given (sub-)problem. The chosen Click basis for our prototype provides the necessary extensibility to integrate solutions for such (sub-)problems.

4.4.5 Application-level Problems

For application purposes, information can be structured so that it maps onto concepts such as knowledge bases, social networking structures or taxonomies of information like videos (e.g., YouTube) or images. Scopes could also carry access policies and, therefore, accessibility to the information structure can be tailored to specific domains of discourse. For example, a YouTube-like service could be represented using a root scope. Under that scope there could be sub-scopes for each user, who could publish its videos in a custom taxonomy of concepts (e.g., funny, science and favourite videos), itself being represented as sub-scopes published under the user's scope. Other examples for sub-scopes could be that of *channels* that represent a curated video collection. Video sharing could be achieved by implementing appropriate access policies for specific scopes. Furthermore, specific information items (e.g., a video) can be published under several scopes.

4.5 Initial Performance Results

We now present initial performance results for our prototype. This evaluation shall not be seen as a final word on performance and scalability of our methodology as such. Instead, we would like to demonstrate that we are able to achieve reasonable performance for an example system that is entirely built on our system design approach. Keeping in mind the outlined possibilities for improving memory management, the current results will provide us with an indication of the expected performance of a final system.

4.5.1 Setup & Experiments

For our experiments, we implement a ping-like application that publishes information items in a given scope towards a set of subscribers under a given dissemination strategy. In a first version, the application advertises the availability of data under a given information identifier and publishes a stream of data once it receives the RV component notification. With that, we emulate dissemination within a channel of *mutable* information, such as for video data. In a second version, the operation advertises a single information item and publishes its data upon receiving a RV component notification. This emulates the case of *immutable* application data. All of the items of information are published under a single scope for simplicity purposes. The application tests the two currently implemented link- and domain-local dissemination strategies. For the first one, we choose a simple setup in which the publisher resides in one machine with the subscriber in the link-locally connected other machine (see Figure 10 top).

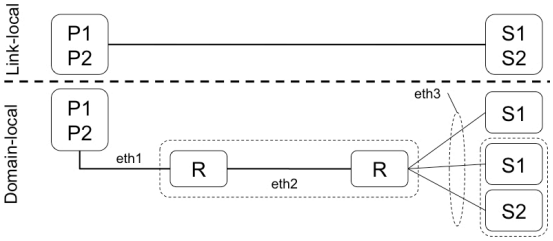

Figure 10: Experimental Setup

The second strategy is tested with a setup of a simple graph of multiple Ethernet segments with two publishers and multiple subscribers, the segments being connected by dedicated nodes (that perform only forwarding, not directly subscribing to the data). The bottom of Figure 10 shows our setup for this strategy. All Ethernet segments run at

100MBit/s in the domain-local and at 1GB/s in the link-local strategy. The nodes in dashed boxes are executed as separate virtual machines within a dedicated desktop.

4.5.2 Results

Let us first discuss the link-local results, as shown in Table 1. In the first scenario, there is a single publisher (P1) in the first and a subscriber (S1) in the second machine. The second scenario adds a subscriber in the second machine, while the third scenario establishes two publisher/subscriber pairs. The first numbers present the results for the mutable data operation, with the immutable operation results in brackets.

We can observe that the link-local strategy reaches its limit at 8.9 MB/s throughput in a single subscriber scenario with the mutable data operation. Adding the second subscriber doubles the total throughput since the copy operation for the second publication is performed on the second machine. The numbers are confirmed with theoe of the two publish-subscribe pairs.

In all scenarios, we observe packet loss at the publisher. This is due to send buffers overflows in the Click Packet Socket Element. In order to corroborate this, we implement a separate ping application in Click that runs directly over raw IP sockets. In this case, the total throughput in the first scenario increases to 26.7 MB/s, while the third scenario yields in a total throughput of 33.7MB/s with a packet loss of 9.9%. Hence, it is apparent that the Click implementation of the Packet Socket Element limits our ability to maximize the available link capacity.

Table 1: Link-local Strategy

Scenario	Average throughput per subscriber (MB/sec)	Total application throughput (MB/sec)	Average Packet Loss (%)
P1->S1	8.9 (2.9)	8.9 (2.9)	3.1 (0)
P1-> (S1, S1)	8.9 (3)	17.9 (5.9)	2.4 (0)
P1->S1, P1->S2	4.4 (1.35)	8.8 (2.7)	3.1 (0)

In the case of immutable data, the application throughput significantly decreases due to the additional RV component signaling for each publication. Although we configure the strategy so that the RV component resides on the publisher's side, it is the amount of internal operations that causes the slowdown of the application throughput. Given the smaller overall throughput, there is no packet loss being observed since the Click send buffer queues do not overflow.

In our next tests, we measure the performance for the domain-local strategy, utilizing the LIPSIN forwarding mechanism. Table 2 shows the results for these tests. We can observe that there is hardly any degradation of throughput per subscriber (first column in Table 2) in the first scenario, compared to the simple link-local scenario. The slight degradation being observed is largely due to the additional forwarding operations in the elements denoted "R" in Figure 10; these elements implement the forwarding on the LIPSIN identifiers. There is, however, a degradation of performance in the second scenario, resulting in halving overall publication throughput. This is due to the additional copy operation being necessary at the second "R" element in Figure 10 and the send limit imposed by the Packet Socket Element of Click (the reason for the slightly higher overall throughput is that the second "R" element is a faster computer than the previous bottleneck, the publisher, resulting in a slightly higher Packet Socket Element limit). Last but not least, the addition of another publisher/subscriber pair only marginally impacts the total throughput on application level. Across all scenarios, packet loss can be observed at the various elements that involve sending over the Packet Socket Element.

Table 2: Domain-local Strategy

Scenario	Average throughput per subscriber (MB/sec)	Total application throughput (MB/sec)	Average Packet Loss (%)
P1->S1	8.2 (2.6)	8.2 (2.6)	10.6 (0)
P1->(S1, S1)	5.6 (2.7)	11.2 (5.3)	34.8 (0)
P1->(S1, S1), P2->S2	3.5 (1.3)	10.5 (3.9)	21.7 (0)

In the immutable mode of operation, the performance similarly decreases as in the link-local case. The similar numbers result from our configuration since also in this case, the RV component for the domain is configured to reside in the publisher's node.

Although it is clear that the Packet Socket Element limits the overall throughput, the difference to our ping over raw IP socket Click application becomes significantly smaller in the domain-local case. Only 11.3 MB/s are transferred from P1 to S1 in the first scenario, when using IP between individual subnets. This number reduces by 50% due to the lack of using IP multicast, i.e., each publication to another subscriber needs to be sent again. This leads to total throughput numbers of 11.3 MB/s, which is similar to our own Ethernet-based dissemination, although entirely implemented over Click rather than native sockets.

What we can see from our results is that our prototype provides a promising performance for link- as well as domain-local strategies. Given the lack of optimization on memory management level and the execution of all operations in user space and over the Click router framework (some of which even run on virtual rather than dedicated machines), these numbers have to seen as a lower limit of what is possible. In the future, we envision larger-scale tests in facilities such as PlanetLab, facilitated through our support for raw IP sockets instead of Ethernet.

5. FACILITATING DELIBERATION

We are now at a stage where we can assemble systems based on a common nucleus, with an early prototype performing at a reasonable performance. What is missing in fully enabling our problem solving framework, introduced

in Figure 1, is to provide an element of deliberation in the assembly and reconfiguration of problem solutions. The following section addresses this missing element. For that, we need to place new requirements on how dissemination strategies for problem solutions are described[8]. We also outline the interplay between reactive and deliberative processes within our framework.

Let us first briefly discuss the necessity for such deliberation. In the presence of various existing cognitive architectures (see [4][5] for examples as well as [28] for a comprehensive overview), the question arises as to what differentiates our framework from these. As pointed out in [29], integrating intelligence into networks faces many challenges, such as the amount of additional traffic created, the potential change of network structure, the highly distributed character of network elements, and, above all, the strict timing requirements that govern most of the operations within a distributed system. Our human-inspired framework, founded on the meta-reasoning architecture of Chapell and Sloman [2], provides a differentiation between *reactive* and *deliberative* processes. We believe that this differentiation allows for more efficiently addressing time constraints by mainly performing reactive processes at high speed, while using deliberative processes only in the background and only when necessary (i.e., based on certain triggers). In addition, the encapsulation into individual solutions provides means for isolating traffic being generated for deliberation purposes while the dissemination strategy approach provides the flexibility in implementation choice that a problem might require.

For a realization of such deliberation, the notion of a *dissemination strategy*, as introduced as our fifth invariant, is central. In the following, we argue that formalizing such strategies enables the reasoning over and reconciliation of conflicts within a distributed environment, providing the missing element of deliberation within our framework.

5.1 An Extended E2E Principle

As a first step, we extend our E2E principle in Section 3.2 by embedding a notion of reconciling dissemination strategies during runtime of the system:

> *A given problem within a distributed system can be implemented through an assembly of sub-problem solutions, whose individual dissemination strategies can be reconciled at runtime towards an alignment with the ones set out by the problem in question.*

This extended E2E principle shifts the focus of compliance from the *alignment* of strategies at design time (through methods of requirements engineering, specification as well as codification through executable code) onto the *reconciliation* of conflicting dissemination strategies at runtime. It is this reconciliation that represents the missing element of deliberation in our problem solving framework. Such reconciliation requires the ability to reason over identified conflicts, possibly leading to performance degradation or malfunctioning, in order to finally comply with the original intention of a problem solution (and its sub-problems that need to be solved in a recursive manner). Or at least, such reasoning will allow for identifying the conflict(s), with the possibility to retreat to the last resort of manual intervention for a possible reconciliation.

This opens up the question as to how we could enable such reasoning ability within the problem solving framework that we outlined. The next section addresses the most crucial steps towards such ability.

5.2 A Need to Formalize Problem Solutions

Our functional model introduces the notion of a dissemination strategy within an information-centric functional model. Such strategies outline how to realize the functions of finding information (rendezvous), building a graph between supplier and consumer of information (topology) and forwarding information along the graph (forwarding). In our design time version of the framework, we expect these strategies to be defined through the various protocol specifications, design documents and eventually prototype implementation that underline a working system.

When moving from such design time defined behavior towards a system that can detect and eventually reconcile conflicts between given strategies at runtime, we need to move from such specification in design documents and code towards a formal specification of such strategies. We see several steps as being crucial in such formalization.

Firstly, there is a need to *capture the concepts* of the information being utilized in the implementation of a strategy. For instance, our current domain-local strategy, based on the LIPSIN approach [27], utilizes individually labeled links within a local network graph. While there is no individual node addressing, a graph of constant-length Bloom filters describes the location of individual subscribers relative to a given publisher. We see the use of *ontologies* [30] at the heart of capturing such knowledge concepts. These enable the definition of concepts and their relations in a machine-understandable representation. Such capturing of concepts is necessary due to the distributed nature of the knowledge being created and the utilization of such knowledge by various parties implementing individual solutions based on these concepts (which in turn is often expressed in various formats). Only an *ontology-based design* approach allows for reconciling potential differences in concepts being used that could lead to conflicts in dissemination strategies.

In order to come to our second step, we interpret the functions in our functional model as a set of defined services for information dissemination. With that, the dissemination strategy defines the realization of these services and it is the part of our functional model that needs

[8] Note that the problem itself is represented by the labels and scopes within our functional model, defining the information structure that is being disseminated to enable the appropriate usage of resources to solve the problem at hand.

formalization. For that, approaches such as BPEL (Business Process Execution Language [31]) or the service ontology of OWL-S (Web Ontology Language [30]) could be applied, utilizing the knowledge concepts as expressed through our first step. Hence, we can move from capturing information concepts to formally *expressing and exposing realizations* of various dissemination strategies.

It is this formal expression of the core functions of our functional model and its underlying dissemination strategy, that allows for moving to the final step, namely that of *reconciling conflicts*. Given the exposure of the knowledge concepts that underlie the implementation of a strategy as well the exposure of the implementation realization itself in a formal manner, reconciliation can be implemented through mediation and reasoning on various levels of the overall system, similar to related work in [32][33].

Many of the techniques required for the representation of domain concepts in the form of ontologies, the expression of service realizations and the mediation of conflicts have been studied in various efforts. While we have not yet addressed this runtime extension in our current prototype, we are confident that our framework provides the right abstractions for such formalization, enabling the missing deliberation within our problem solving framework.

5.3 Bringing The Pieces Together

The envisioned runtime extension presented in the previous subsections is the missing piece towards implementing our problem solving framework of Figure 1. The functional model, with its core functions and the dissemination strategy that outlines their implementation, provides the common nucleus for solving individual problems within a distributed system. The end-to-end principle, formulated as a design as well as runtime version, enables the assembly of these individual solutions towards a complex distributed system. It is the move from design to runtime that addresses the support for deliberative processes, as outlined in Figure 1; processes that implement deliberation over the goals that are formulated in the process' formal strategy.

One aspect of such deliberation may be that of *fault detection and mitigation*. Here, crucial information is disseminated from highly optimized reactive processes over specialized trigger components towards deliberative processes. The trigger components are responsible for aggregating information from the reactive processes, triggering events towards the deliberative processes based on defined thresholds. Examples for such trigger components could be elements responsible for link availability detection or throughput measurement. Deliberative processes, acting upon such trigger events, could implement solutions for detecting faulty links, in turn leading to reconfiguration of the topology function of underlying reactive processes. It is the formalization of the dissemination strategies of the various reactive processes as well as trigger components that enables the reasoning within the deliberative processes towards a defined long-term goal, such as resilience.

Another possible deliberation is that of *reconfiguring the assembly* of problem solutions itself, effectively changing the boundaries of dissemination of underlying reactive processes. This could lead to opportunities to optimize the operation of the system by changing boundaries of communication and their underlying technologies. Other usages, such as outlined in [1], are optimized application support or intrusion detection. We leave the formulation of such usages within our framework for a future discussion.

6. ENABLING ARCHITECTURES

We have arrived at a point at which we have shown the elements of our framework as well as the realization and evaluation of a particular artifact that we are able to create with it. But the true generality of this framework comes from its element of enabling architectures, such as proposed in related work or existing in the Internet.

We realize that a wider architectural discussion is necessary to more fundamentally address the enabling nature of our approach. In this paper, we can only touch upon this issue. We do so by briefly discussing how some recently proposed architectural efforts could be realized with our proposed functional model by comparing the fundamental concepts used in each of them.

Within Content-Centric Networking (CCN [34]), so-called *interest* requests are disseminated within a local domain (currently via broadcast) while being tunneled to remote domains in the case that requests cannot be fulfilled locally. This determines the return path for any information provider to reply to the original interest request of a subscriber. Functionally, this resembles a particular dissemination strategy for the functions of finding, topology, and forwarding in our model. CCN foresees the implementation of varying strategies for forwarding (interest) requests within a *strategy layer* [34]. At the moment, however, only local broadcast with domain-level tunneling is documented. A form of formal expression of such strategy is not envisioned but can easily be realized as an addition to the outlined strategy layer mechanism. Caching plays an important role within CCN with caches in the interest forward path being able to reply to requests if the information is available in the cache. From a functional point of view, this can be represented as a solution to a sub-problem in alignment with our framework. The most notable difference in CCN is the usage of a hierarchical naming scheme, akin to the Domain Name Service (DNS), for the labeling of information. Within our functional model, however, this can be seen as a particular (hierarchical) implementation of an application-level naming scheme. The work in [35], for instance, demonstrates how to map such hierarchical names on a scoping/labeling concept as it is used in our functional model, including the securing of the information content. It is important to note that CCN aims at providing a different internetworking layer to the Internet. It is not a general system design approach. Hence, any realization of a distributed solution has to be seen in combination with a

possible IP underlay (CCN is specifically assumed to evolve alongside IP networks) and naming-based overlays.

Similar to CCN, the PSIRP project [36] aims at developing a new internetworking layer for the Internet. Its currently formulated concepts largely match the invariants of the functional model of our framework. Scoping and labeling of information is used together with functions for finding information and forwarding the actual data along constructed delivery graphs. Although various levels of rendezvous are envisioned [37], only a global rendezvous solution is specified. The work in [22] and [23] outlines a layering concept that resembles similarities with our approach, albeit with little indication of its implementation within the PSIRP architecture. Given these similarities and the wider system design approach of our work, we see interesting possibilities to extend the PSIRP work towards a larger system vision with a clear model for recursive layering and its implementation.

The Data-Oriented Network Architecture (DONA [38]) shares many similarities with our functional model. While scoping is not directly supported through a dedicated (scope) identifier, the P:L nature (P identifying the principal of a transaction and L the information itself) of DONA identifiers resembles a scoping nature that can easily be reproduced with our functional model. The notion of *data handlers* [38] implements a functional scoping for the rendezvous functionality within DONA with each data handler being responsible for data items that are advertised towards it. Topology and forwarding functions overlay on the IP routing fabric. With that, we see DONA being easily implementable through our functional model.

An almost obvious question is that of how IP-based architectures fit into our system design thinking. The obvious difference between an IP-based architecture and our framework is the information-centrism that underlies our work; something that is in apparent contrast to the endpoint-centric notion of IP. However, work in [39] has shown that socket-based communication based on endpoint identifiers can easily be implemented over an information-centric network (here the PSIRP architecture). The trick is to interpret IP addresses as a scoped data structure over which to apply on information-centric service model. We recognize that there is more to enabling an IP-based architecture than emulating the socket interface at the internetworking layer. Management as well as service interfaces need to be enabled via an information-centric approach. However, the authors in [40] argue that many of these interactions are in fact information-centric, proposing HTTP as the (new) waist of an information-centric architecture. Only a thorough architectural argumentation can provide clarity if this assertion is indeed true. But it can be seen as an indication that assembling IP from individual (information-centric) problem solutions could be feasible.

As another branch of research, efforts such as proposed by Strulo et al. [41] or the meta-routing efforts presented in [16] envision the design and configuration of networks based on mathematical concepts. Its decomposition into sub-problems as well as the mathematical approach to formulate dissemination strategies bears similarities with the functional approach that we present in this paper. However, we leave the discussion to merge our framework with a mathematical approach to design for future work.

7. CONCLUSIONS

In this paper, we proposed a nucleus for building layer-less complex distributed systems, following a problem solving approach at its core. At the abstract level, the communication system is seen as an information dissemination medium for problem solutions. These solutions are being recursively assembled towards a large-scale system and reflect functions in a networked environment. Such functional view allows for the definition of an end-to-end principle in which adherence is reflected by the proper alignment of strategies. It is the grander goal of our framework that malfunctioning can be reasoned over and pinned down to particular parts of the system through identifying the strategies that are in conflict.

This paper provided a first insight by outlining the framework and an example for an artifact that is based on it. What this paper only started is the architectural argumentation that supports this methodology; an argumentation that underpins the evidence presented in this paper with the understanding why it could indeed be the right nucleus for many architectures to build upon.

We assert, however, that our approach provides the right abstractions for this ambition. The exposure of a dissemination strategy per problem solution allows for formulating and reasoning over goals and strategies, while providing a simple enough model for the individual parts, each of which can be optimized in its individual operation. This combination of high-level abstractions that enable the implementation of deliberative processes as well as the optimization of reactive processes aligns our view with that of human-inspired meta-reasoning architectures – it is this alignment that we see at the heart of implementing the vision of the Knowledge Plane.

REFERENCES

[1] D. Clark, C. Partridge, J. C. Ramming, J. Wroclawski, “A knowledge plane for the internet”, Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM, 2003

[2] J. Chappell, A. Sloman, “Natural and artificial meta-configured altricial information-processing systems”, Invited contribution to a special issue of The International Journal of Unconventional Computing, Vol 2, Issue 3, 2007

[3] S. J. Russell, P. Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition, Pearson Education, 1998

[4] M. Wawrzoniak, L. Peterson, T. Roscoe, “Sophia: An Information Plane for Networked Systems”, Proc. of the 2nd Workshop on Hot Topics in Networks, Nov. 2003

[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, “A Clean Slate 4D Approach to Network Control and Management”, ACM

SIGCOMM Computer Communication Review, Vol. 35, No. 5, Oct. 2005

[6] The OpenFlow Switch Consortium, available at http://www.openflowswitch.org/, 2010

[7] S. Quirolgico, K. Mills, D. Montgomery, "Deriving Knowledge for the Knowledge Plane", National Institute of Standards and Technology, June 2003

[8] R. Chadha, G. Lapiotis, S. Wright, "Policy-Based Networking", IEEE Network special issue, Vol. 16, No. 2, March/April 2002

[9] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", Scientific American Magazine, May 2001

[10] L. Feigenbaum, "The Semantic Web in Action", Scientific American, May 2007

[11] R.W. Thomas, L.A. DaSilva, A.B. MacKenzie, "Cognitive networks", in: Proceedings of the First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, Baltimore, MD, USA, November 8–11, 2005

[12] ANA project at http://ana-project.org, 2010

[13] BioNets project at http://www.bionets.eu, 2010

[14] N. Agoulmine, S. Balasubramaniam, D. Botvich, J. Strassner, E. Lehtihet, D. Donnelly, "Challenges for Autonomic Network Management", 1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE), 2006

[15] N. Foster, R. Harrison, M. Meola, M. Freedman, J. Rexford, D. Walker, "Frenetic: A High-Level Language for OpenFlow Networks", Proceedings of PRESTO workshop in conjunction with ACM CoNext conference, November 2010

[16] P. J. Taylor, T. G. Griffin, "A model of configuration languages for routing protocols". PRESTO workshop in conjunction with ACM SIGCOMM, August 2009

[17] X. Chen, Y. Mao, Z. M. Mao, K. van der Merwe, "Declarative Configuration Management for Complex and Dynamic Networks", Proceedings of ACM CoNext, 2010

[18] K. Sollins, "Designing for scale and differentiation", Proceedings of the ACM SIGCOMM FDNA workshop on Future directions in network architecture, October 2003

[19] T. Roscoe, "The End of Internet Architecture", Proceedings of HotNets, 2006

[20] J. Day, "Patterns in Network Architecture: A Return to Fundamentals", Pearson Education Publisher, 2008

[21] J. Touch, Y. Wang, V. Pingali, "A Recursive Network Architecture", ISI Technical Report ISI-TR-2006-626, October 2006

[22] M. Särelä, T. Rinta-aho, S. Tarkoma, "RTFM: Publish/Subscribe Internetworking Architecture", Proceedings of ICT Mobile Summit 2008, July 2008

[23] N. Fotiou, G. C. Polyzos, D. Trossen, "Illustrating a Publish-Subscribe Internet Architecture", Proc. of Future Internet Architectures: New Trends in Service Architectures, 2009

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. F. Kaashoek, "Click modular router", ACM Transactions on Computer Systems (TOCS), Vol. 18 Issue 3, Aug. 2000

[25] "The iGraph library for complex network research", available at http:// igraph.sourceforge.net, 2010

[26] P. Nikander, B. Nyman, T. Rinta-aho, S. Sahasrabuddhe, J. Kempf, "Towards Software-defined Silicon: Experiences in Compiling Click to NetFPGA", 1st NetFPGA developers workshop, September 2010

[27] P. Jokela, A. Zahemszky, S. Arianfar, P. Nikander, C. Esteve, "LIPSIN: Line speed Publish/Subscribe Inter-Networking", ACM SIGCOMM, August 2009

[28] T. Dietterich, P. Langley, "Machine Learning for Cognitive Networks: Technology Assessment and Research Challenges". In "Cognitive Networks: Towards Self-Aware Networks", chapter 5, pg. 97, edited by Qusay H. Mahmoud, Wiley-Interscience, 2007

[29] P. Langley, J. E Laird, S. Rogers, "Cognitive architectures: Research issues and challenges", in Cognitive Systems Research, vol. 10, issue 2, Elsevier B.V., pgs: 141-160, 2009

[30] W3C, "OWL-S: Semantic Markup for Web Services", available at http://www.w3.org/Submission/OWL-S/, 2004

[31] OASIS, "Web Services Business Process Execution Language Version 2.0", available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007

[32] K. Sycara, R. Vaculín, "Process Mediation of OWL-S Web Services", in Advances in Web Semantics I, Springer Verlag, 2009

[33] A. Mocan, E. Cimpian, "An Ontology-Based Data Mediation Framework for Semantic Environments", Journal on Semantic Web & Information Systems, Vol. 3, No. 2, 2007

[34] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, R. L. Braynard, "Networking Named Content", Proceedings of ACM CoNext, 2009

[35] W. Wong, P. Nikander, "Secure Naming in Information-Centric Networks", Proceedings of ReArch workshop in conjunction with ACM Conext, 2010

[36] PSIRP project at http://www.psirp.org, 2010

[37] S. Tarkoma, D. Trossen, M. Sarela, "Black Boxed Rendezvous Based Networking", Proceedings of MobiArch 2008 workshop at ACM SIGCOMM, August 2008

[38] T. Koponen et al., "A Data-Oriented Network Architecture", ACM SIGCOMM, 2007

[39] G. Xylomenos, B. Cici, "Design and Evaluation of a Socket Emulator for Publish/Subscribe Networks", Future Internet Symposium, September 2010

[40] L. Popa, A. Ghodsi, I. Stoica, "HTTP as the Narrow Waist of the Future Internet", Proceedings of HotNets, 2010

[41] B. Strulo, N. Walker, M. Wennink, 'Lyapunov Convergence for Lagrangian models of Network Control', T. Chahed and B. Tuffin (Eds.): NET-COOP 2007, LNCS 4465, pp.168-177, 2007