2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems

TARL: Modeling Topology Adaptations for Networking Applications

Michael Stein Alexander Frömmgen Roland Kluge Frank Löffler
Andy Schürr Alejandro Buchmann Max Mühlhäuser
TK / DVS / ES, TU Darmstadt, Germany
{stein, muehlhaeuser}@tk.tu-darmstadt.de
{froemmgen, buchmann}@dvs.tu-darmstadt.de
{roland.kluge, andy.schuerr}@es.tu-darmstadt.de

ABSTRACT

Many networking applications implement topology adaptations to cope with network dynamics. Related work focuses on the specific application, lacking a general model for topology adaptations.

In this paper, we analyze 14 topology adaptations from two different application domains. Based on the derived characteristics, we propose a general topology adaptation model. We present the *Topology Adaptation Rule Language* (TARL) to specify topology adaptation logic following this model. We discuss the execution of TARL rules for two application domains as well as how our model enables reasoning and optimizations on topology adaptations. For the evaluation, we developed a TARL runtime environment as a reusable topology adaptation framework. TARL simplifies the development of topology adaptations and is able to express 13 of the analyzed algorithms.

1. INTRODUCTION

Networking applications are facing rapidly changing environments. Many of these networking applications, e.g., from the domain of *Video Streaming Overlays* (VSOs) or *Wireless Sensor Networks* (WSNs), adapt their connections between communication partners, the so-called *topology*, in a decentralized way to cope with changes. VSO applications maintain a topology for content distribution (see Figure 1a). Different types of topologies (e.g., tree [29] or mesh [21]) are better suited depending on the current environment [32]. Many proposed adaptations optimize the topology at runtime, e.g., to improve quality of experience [30] or balance the load [29]. In today's Internet of Things, a large number of wireless sensors measure environmental data [34], forming a data collection topology (see Figure 1b). Wireless sensor devices have very limited hardware resources [22]. Topology adaptations enable these devices to reduce their transmission range, thus leading to better energy conservation [25] or throughput [19]. Topology adaptations are further applied, e.g., for event dissemination in dynamic virtual realities [18], load balancing in search overlays [16], or connectivity maintenance in peer-to-peer networks [20].

We observe that, although topology adaptations are crucial for the performance of many networking applications, their development and presentation in the networking community does not follow a well-defined model. Explicit adaptation models as established in the self-adaptive community are not applied. The adaptation logic is often tightly integrated into the application. This makes it hard to understand and compare existing topology adaptations. Reusing existing concepts and implementations is difficult. Thus, the lack of a well-defined model substantially complicates the development, implementation, evaluation and maintenance of topology adaptations in networking applications.

In this paper, we present a general model for the specification and execution of topology adaptations in networking applications. We derive the requirements of such a model based on 14 existing topology adaptations from two application domains. We observe that these adaptations conduct three main steps that are compliant with control loop concepts of self-adaptive systems:

1. The application *monitors* (one or more logical) topologies, relevant events and metrics.
2. The application decentrally *decides* if an adaptation is required. This decision is based on graph patterns in the monitored topology.
3. The application *executes* the adaptation as a sequence of simple operations.

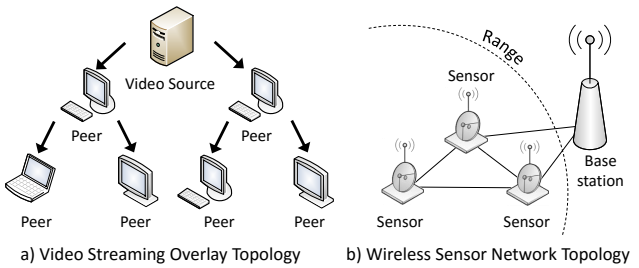

Figure 1: Example of two networking application domains that adapt their topologies at runtime

SEAMS'16, May 16-17, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897053.2897061

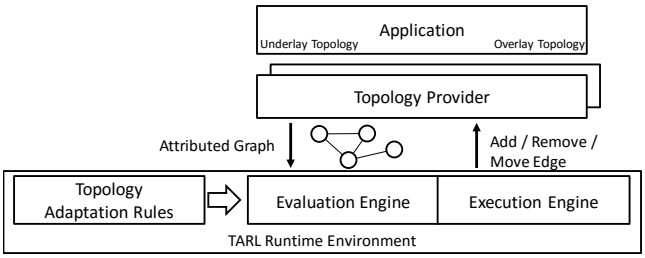

Figure 2: Overview of the topology adaptation loop for networking applications

We provide the developer with a model for these steps. The developer describes the graph patterns, conditions and events that trigger a topology adaptation. The conducted adaptation is specified as a sequence of simple graph operations. We propose TARL, a *Topology Adaptation Rule Language*, as corresponding specification language. TARL rules can be interpreted in a runtime environment (see Figure 2) or used for the generation of platform-specific code. This approach separates the topology adaptation logic from the application's implementation, thereby fostering the reuse of topology adaptations across application domains, enabling rapid prototyping of topology adaptations and easing the maintenance of the adaptation logic. The evaluation with 14 adaptations from the VSO and the WSN domain shows that TARL provides a convenient specification of the adaptation logic. Moreover, we demonstrate that executing rules in our implemented runtime achieves similar performance for a VSO application scenario as an existing, tightly integrated adaptation implementation.

This paper provides the following contributions:

1. We identify the characteristics of topology adaptations of networking applications.
2. We present a model and a specification language for expressing the topology adaptation logic in networking applications.
3. We show how this language expresses adaptation logic and demonstrate the execution of rules in our runtime environment.

The remainder of this paper is structured as follows: First, Section 2 derives the characteristics of topology adaptations, which serve as requirements for the developed model and TARL in Section 3. Section 4 discusses the execution and optimization of TARL rules. Section 5 provides the evaluation, followed by a discussion of related work (Section 6) and the conclusion (Section 7).

2. ADAPTATION ANALYSIS

This section explores two exemplary application domains of topology adaptations: VSOs and WSNs. Having inherently different target environments and optimization goals, these domains will then serve for identifying general characteristics of topology adaptations.

2.1 Application Domains

In Video Streaming Overlay (VSO) applications, one or multiple servers provide a video stream. The participating devices, so-called *peers*, play the video stream and distribute the stream further in the network. Therefore, the VSO application constructs a topology (see Figure 1a). VSOs are highly dynamic regarding environmental conditions, and the set of participants and their usage pattern [32]. Depending on the current circumstances and requirements, different types of topologies (e.g., tree [29] or mesh [21]) lead to better application performance [32]. Topology adaptations are necessary to cope with these dynamics at runtime. In particular, the topology should be maintained to ensure high resilience against sudden peer failures [24], and to improve quality of experience [30] or load balancing [29].

Wireless Sensor Networks (WSNs), e.g., in the Internet of Things, consist of a large number of sensor devices that report monitored data to a central base station [34]. Due to the limited transmission range of sensor devices, the data is forwarded over multiple hops through the topology (see Figure 1b). Sensor devices typically have only limited capabilities and no external power supply, which makes selecting efficient communication links a prime goal of topology adaptations in WSNs. Executing topology adaptation, each sensor device selects a subset of neighbors from the given set of physical neighbors and restricts the communication to these logical neighbors [31]. This approach allows for shrinking the transmission range of the devices, thereby particularly improving energy conservation [25]. Also, it has been shown that topology adaptations in wireless networks can be used to increase the throughput [19].

2.2 Adaptation Characteristics

Having analyzed various topology adaptations from the domains of VSOs [21, 24, 26, 29, 30] and WSNs [12, 19, 25, 31, 33], we derive the following five major characteristics (C1–C5) of topology adaptations in networking applications.

C1 Adaptation Loop: We observe that topology adaptations conduct three main steps that are compliant with control loop concepts of self-adaptive systems, such as the MAPE-K loop [13]. The application *monitors* the topologies and additional relevant events and metrics. Based on this, the adaptation algorithm *decides* if an adaptation is required. Finally, the application *executes* a set of topology operations.

C2 Decentralization: Networks may have a huge size [24]. For scalability reasons, topology adaptations are executed in a *distributed* way. In other words, each device locally executes the adaptation loop and modifies its surrounding topology.

C3 Graph-based Decisions and Operations: A topology may be modeled as a graph whose nodes and edges represent devices and communication links, respectively. For adaptation decisions, graph patterns are analyzed. These graph patterns include the connectivity between nodes as well as properties of nodes and edges (e.g., latency). We observe that the conducted graph operations are very simple. For example, kTC [25] disables dispensable communication links from a WSN (i.e., kTC *removes* edges). Other topology adaptations establish new communication links (i.e., they *add* edges), e.g., to increase connectivity of the network [20]. Often, one link is removed and added in an atomic step (i.e., an edge is *moved*).

C4 Multiple Topologies: The analyzed networking applications are based on at least two topology layers: the *underlay* and the *overlay*. While the underlay reflects physical connectivity between the devices, the overlay describes logical communication patterns. A particular topology adaptation may affect one or more of these topologies. For example, the TRANSIT video streaming application [32] maintains a basic neighborhood overlay and multiple *flow* topologies to deliver a video stream in various encodings. As topologies have complex dependencies [27], many topology adaptations consider multiple topologies for the decision and the execution.

C5 Limited Local View: Collecting topology information is expensive and does not scale well. Following the concept of local algorithms [28], a device has only a restricted *local view* of the topology. Sensor devices typically have limited memory [22], which makes storing global topology information infeasible. High dynamics in VSO applications make it difficult to maintain a global view of the topology [32].

3. MODEL AND LANGUAGE

This section introduces the concepts and the syntax of *TARL*, the first **T**opology **A**daptation **R**ule **L**anguage for networking applications. In contrast to existing adaptation rule languages, TARL is well-suited to express adaptation logic for networking applications with their characteristics (C1–C5) observed in Section 2.2.

TARL describes the topology adaptation logic in a rule-based manner, separating monitoring, adaptation logic and execution of the adaptation (C1). Each device executes these rules in a decentralized way (C2). A TARL rule consists of three subsequent parts. A *preamble* may contain a number of constant declarations and selector declarations to simplify multiple instantiations of a rule. The *condition* specifies if the *execution* part of the rule should be triggered.

3.1 Graph Model

We describe the communication system as an *attributed graph* $G(V, E)$ consisting of *node*s V, representing the devices, and directed *edge*s E, representing the communication links between the devices (C3). The node that represents the current device is called *self*. Nodes and edges may have a set of *attribute*s. To model coexistent topologies (C4), each edge e is labeled with a *topology identifier* attribute $\text{tid}(e)$. A *topology* $T(t)$ with topology identifier t is the sub-graph of G induced by the set of all edges having topology identifier t. Note that a node may be part of multiple topologies, whereas an edge is contained in exactly one topology. In a running system, the graph is monitored by a set of *topology providers*. Each provider is responsible for one or more topologies and contributes to G those edges that belong to these topologies. Our model considers three *topology modification operations* (C3): *add*, *remove* and *move edge*.

3.2 Sample TARL Rule

Listing 1 shows a TARL rule. The purpose of this rule is to add an edge from *self* to another node *n1* in topology *T2* (line 11) if the following conditions are fulfilled: In topology *T1*, node *n1* is the neighbor of *self* with the maximum edge weight that has an additional edge (lines 2–7). Additionally, *self* has an edge *e2* to a neighbor *n3* in topology *T3*. The edge between *self* and *n1* is only added if *n1* has no neighbor in topology *T2* (line 8–9).

```
1  filter(
2    max(
3      join(
4        match(TP, T1, self <- e0 -> n1,
5          n1 <- e1 -> n2),
6        match(TP, T3, self <- e2 -> n3)),
7      e0.weight),
8    count(
9      match(TP, T2, self - e3 -> n4)) = 0)
10 execute every match:
11   at(self, TP, T2) add neighbor(n1)
```

Listing 1: Sample TARL rule

3.3 Condition Specification

The condition declaratively specifies at which locations in the graph G the rule may be applied. Its fundamental concepts are topology patterns and matches: A *topology pattern* P is a graph of limited size (C5) consisting of node and edge variables, which are placeholders for nodes and edges of a graph. A node variable may appear in multiple patterns; an edge variable may appear in one pattern only. In TARL, variables are identical if they bear the same name. In general, a *match* m is a mapping from a set of node (edge) variables to the nodes (edges) of G that preserves source and target of edge variables. In particular, a *match m of a topology pattern P in a graph G wrt. a topology T* is an injective mapping from the node (edge) variables of P to the nodes (edges) of G that preserves source and target of edge variables and that maps edge variables only to edges in topology T.

The condition is a nested expression whose evaluation results in a set of matches. In the following, we explain the most important expressions that may appear inside the condition. For simplicity, we denote expressions that evaluate to sets of matches as well as sets of matches with M.

match(tp, t, P): A *match expression* evaluates to the set of all matches of pattern P in the graph G wrt. topology $T(t)$ provided by topology provider tp. Different node placeholders in the pattern relate to distinct nodes in the monitored graph G. In the example in Listing 1, the first match expression evaluates to all matches of two chained edges *e0*, *e1* in the *T1* topology provided by *TP*. Note that *e3* bears an arrow head to specify a direction, while *e0* is an undirected edge.

join(M_a, M_b): A *join expression* joins the sets of matches resulting from the expressions M_a and M_b, where M_a and M_b may be any expressions that evaluate to a set of matches. The join expresses relationships and dependencies between multiple topologies (C4). We say that two matches m_a and m_b are *compatible* if each node variable[1] that is in the domain of both matches is mapped to the same node in G, i.e., $m_a(x) = m_b(x) \forall x \in \text{dom}(m_a) \cap \text{dom}(m_b)$. The resulting match set M_{ab} contains a match m_{ab} for each compatible pair of matches m_a, m_b from $M_a \times M_b$, where m_{ab} is defined for all variables in $\text{dom}(m_a) \cup \text{dom}(m_b)$ and the restriction of m_{ab} to the domain of m_a (m_b) is identical to m_a (m_b); i.e.,

[1]Note that by definition, there are no common edge variables in $\text{dom}(m_a)$ and $\text{dom}(m_b)$.

$m_{ab}\restriction_{\mathrm{dom}(m_a)} = m_a$ and $m_{ab}\restriction_{\mathrm{dom}(m_b)} = m_b$[2]. Note that the resulting match m_{ab} might not be injective. In Listing 1, the distinct node variables *n1* and *n3* may be mapped to the same node in the joint match.

filter(M, f_B): A *filter expression* filters a set of matches using a Boolean-valued function f_B. As before, M may be any expression that evaluates to a set of matches. The function f_B may contain node and edge variables, and references to their attributes. We say that *match m fulfills the Boolean function f_B* if and only if each node and edge variable in f_B is in the domain of m and f_B evaluates to true when replacing all node and edge variables x of f_B with $m(x)$. The result of evaluating a filter expression is the set of all matches $m \in M$ that fulfill f_B.

min(M, f_R), max(M, f_R): *Min* and *max expressions* filter a set of matches, using a real-valued function. As before, M may be any expression that evaluates to a set of matches. The value $f_R(m)$ for function f_R and match m is defined if each node and edge variable in f_R is in the domain of m, and it is calculated by replacing all node and edge variables x of f_R with $m(x)$. The result of *min* (*max*) is the set of all matches $m \in M$ whose value $f_R(m)$ is minimal (maximal) compared to all other matches in M.

count(M): The *count expression* takes a set (of matches) and returns its size. In contrast to the previous expressions, *count* is typically used *inside* real-valued or Boolean-valued functions to express dependencies between topologies.

Each of the other statements and therefore each match set M might optionally be labelled. The count expression takes an optional label as parameter. In case this parameter is provided, the count expression returns the number of matches that are *compatible* with the labelled match set.

3.4 Execution Specification

The *execution part* is triggered for each match resulting from the condition. Note that this implies that the execution part might be executed multiple times for different matches. It consists of a sequence of *add edge*, *remove edge* and *move edge* operations, where *move* reassigns the source or target of an edge. These operations may refer to node and edge variables of the match in the condition. Even though the effect of the move operation is equivalent to removing and adding an edge, we observed that this operation is frequent in applications and should thus be modeled explicitly.

4. FRAMEWORK

In the following, we present our prototype runtime for Java applications, sketch possible WSN implementations and envision optimizations enabled by TARL.

4.1 Runtime Environment

The adaptation logic, specified as TARL rules, can be deployed and executed in a runtime environment. The selection of an appropriate runtime depends on the application domain and the device type. VSO applications, for example, are typically executed on commodity hardware and implemented in languages such as Java. WSN applications have to deal with limited resources and are, therefore, often implemented in C. The runtime environment must reflect this.

[2]With $f\restriction_{\mathrm{dom}(g)}$, we denote the restriction of f wrt. the domain of g.

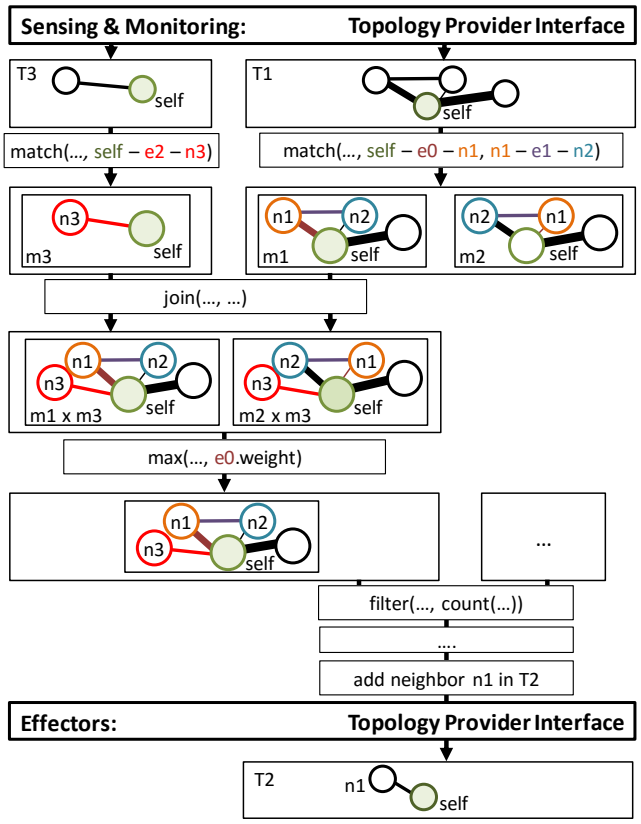

Figure 3: Processing example of Listing 1

Java Runtime for VSOs: Video streaming peers provide the processing capabilities for an interpreter-based rule evaluation. We implemented a Java-based TARL interpreter and a runtime environment that realizes the architecture shown in Figure 2. We integrated the runtime into the simulation platform Simonstrator [23].

Each peer executes an instance of the runtime. A topology provider interface provides a graph-based view of the nodes' local topologies. During the initialization, the runtime parses the rules and resolves dependencies. In particular, it subscribes to all relevant topology updates. A pattern matcher in the evaluation engine detects patterns as specified in the TARL rules.

Sample Rule Evaluation: A concrete evaluation of Listing 1 in our runtime is shown in Figure 3. Based on the local view of the topology providers, the condition is evaluated step by step. Figure 3 illustrates the flow of the intermediate results. Finally, an edge to node *n1* is added to topology *T2* through the topology provider.

Execution in WSNs: For resource-restricted devices, an interpreter-based approach is infeasible. For example, sensor devices based on the *Telos platform* [22] have only 10kB of RAM. We are currently developing a TARL compiler that generates optimized C-code for specific target environments.

4.2 Reasoning and Optimization

Computation: We envision to use optimization techniques, such as pushing filter statements. In Figure 3, for example, the concrete match for *T3* is not required, and a materialization of all combinations in the *join* is not neces-

sary. However, the match term must not be removed, as *n1* is only added if *self* has at least one neighbor in *T3*.

Monitoring: We envision various automatic monitoring overhead reductions with TARL. The rule in Listing 2, for example, is transformed from a representation that requires 2-hop knowledge to a representation requiring only 1-hop knowledge. Moreover, distributing the evaluation and execution of rules in the network may lead to further optimizations. Instead of an expensive communication for an extended local view monitoring, we could exchange intermediate results of the evaluation.

```
/* 2-hop view required */
match(self - e0 -> n1, n1 - e1 -> n2) ...
  at (self) add neighbor n2

/* semantically equal 1-hop rule */
match(n1 - e0 -> self, self - e1 -> n2) ...
  at (n1) add neighbor n2
```

Listing 2: Two semantically equal rules

5. EVALUATION

This section evaluates the proposed model for topology adaptations in networking applications. We present a case study with existing topology adaptations and discuss the expressiveness, the generalizability of the language features, the derived characteristics and our implemented runtime.

5.1 Case Study

For the case study, we present TARL examples to model the *Yao* topology adaptation from the WSN domain and the *low delay jump* from the VSO domain, as these two examples provide a good illustration of TARL's concepts.

Example Yao: In the *Yao* topology adaptation [33], each device divides the plane into k equal-sized cones, where each cone is defined by the corresponding angle to the x-axis. Then, the device adds the closest neighbored device of each cone from the *UDG* topology to the *logical* topology. The concrete model as TARL rule depends on the available monitoring information. Listing 3 shows an example where each node knows its own coordinates and the coordinates of its neighbors in the *UDG* topology. The rule uses a selector for each cone and matches all nodes in this cone by conducting filtering. For this purpose, the coordinates are transformed into angles. From the resulting match, the node with the lowest distance is added. For application scenarios where the angle is directly available, a more compact rule would be possible.

```
define (k, 6)
selector curCone [1..k]

min(
  filter(
    match(TP, UDG, self - e1 -> n1),
      atan2(- n1.y + self.y, -n1.x + self.x)
       + 180 >= (360 / k) * (curCone-1) and
      atan2(- n1.y + self.y, -n1.x + self.x)
       + 180 < (360 / k) * curCone),
  e1.length)
execute every match:
  at (self, TP, LogicalTopology)
    add neighbor (n1)
```

Listing 3: TARL rule for the *Yao* topology adaptation

Example mTreebone LDJ: *mTreebone* is a tree/mesh-based VSO application [30]. Changing environmental conditions and node dynamics cause non-optimal substructures in video streaming trees. One crucial aspect for the quality of experience as perceived by the user is the latency of stream delivery. The *low delay jump* (LDJ) tries to minimize latency by reducing the depth of the streaming tree. For this purpose, nodes replace their parents by neighbors with a lower tree depth. Listing 4 shows the LDJ adaptation as TARL rule. Join and filter statements allow a compact representation of the adaptation logic.

```
filter(
  join(
    match(TP, Tree, n1 - e0 -> self),
    filter(
      match(TP, Neighborhood, n2 - e1 -> self),
      n2.Tree.treeDepth != -1)),
  n1.Tree.treeDepth > n2.Tree.treeDepth)
execute every match:
  at (self, TP, Tree) move neighbor (n1, n2)
```

Listing 4: TARL rule for *mTreebone's* LDJ adaptation

Language Expressiveness: For a comprehensive evaluation of the expressiveness, we modeled the considered algorithms from Section 2 as TARL rules. Table 1 provides a structured overview[3].

TARL is able to specify 13 out of the 14 analyzed algorithms with between 7 and 42 lines of code. TARL cannot express LMST [19], as this would require matching paths of arbitrary length. Even though TARL could be extended to support LMST, we discarded this extension as it would require adding a feature solely for LMST.

The *filter-*, *max/min-* and *selector*-features are used repeatedly in both domains. Although WSNs exhibit multiple topologies, the corresponding rules' conditions only rely on one of these topologies. Thus, *joins* are only used in the VSO domain. The usage patterns of the features show that TARL provides the appropriate level of abstraction.

The used graph attributes in the rules provide a comprehensive overview of the regarded metrics and assumptions of the developer. We found that most rules use only a single graph attribute.

Characteristics: The modeled rules further confirm the identified characteristics. We find that the conditions of all analyzed algorithms use a local view of at most 2 hops on at least 2 topologies. The execution specifications only rely on the graph operations (add, remove and move edge).

5.2 Runtime

To show that the execution of TARL rules in our runtime leads to the same effects as traditional tightly coupled approaches, we used TRANSIT [32], an existing VSO application. This application was extended by a topology provider. We specified TRANSIT's LDJ adaptation [24], which is a modification of mTreebone's LDJ outlined before, as TARL rule. Recall that LDJ aims at minimizing the depth of the streaming tree. Figure 4 shows a comparison of the average tree depth between TRANSIT without topology adaptations, TRANSIT with activated LDJ, and TRANSIT optimized by our runtime based on our TARL specification of LDJ. For this comparison, we simulated a

[3]All modeled TARL rules are available at http://www.dvs.tu-darmstadt.de/tarl

Table 1: Overview of used TARL features for existing topology adaptations from different domains

Domain	Algorithm	Modeled	Lines of Code	Filter	Join	Max/Min	Selector	Topologies	Local View Size	Add	Remove	Move	Attributes
VSO	Chunkyspread [29]												
	Load Optimization	√	18	√	√		√	≥ 3	1			√	n.trgetLoad
	Latency Optimization	√	25	√	√		√	≥ 3	1			√	n.trgetLoad
	Distributed Market Model [21]	√	42	√	√	√	√	≥ 2	1			√	n.uploadBw,...
	DCO [26]												
	Provider Selection	√	10	√		√		2	1	√			n.uploadBw
	Requester Selection	√	7			√		2	1	√			n.uploadBw
	mTreebone [30]												
	Low Delay Jump	√	9	√	√			2	1			√	n.Tree.treeDepth
	High Degree Preemption	√	20	√	√			2	2	√	√	√	n.Tree.treeDepth
	TRANSIT TopT [24]												
	Low Delay Jump	√	21	√	√	√	√	≥ 2	1			√	n.Flow.treeDepth
	High Degree Preemption	√	41	√	√	√	√	≥ 3	2	√	√	√	n.Flow.treeDepth
WSN	kTC [25]	√	17	√				2	2		√		e.weight
	Gabriel [31]	√	9	√				2	2		√		e.length
	Relative-Neighborh. [12]	√	9	√				2	2		√		e.length
	Yao [33]	√	13	√		√	√	2	1	√			e.length, n.x, n.y
	LMST [19]	no											

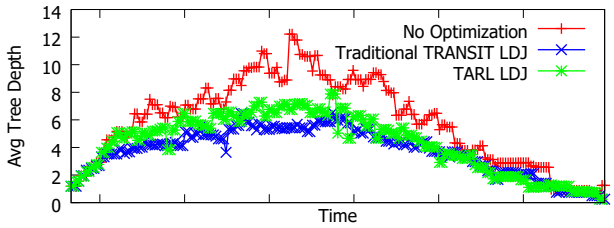

Figure 4: Comparison of the average tree depth in TRANSIT with the LDJ adaptation

streaming scenario with 1000 peers that join and leave the network on a non-regular basis. Both topology adaptations achieve a similar reduction of the tree depth compared to the configuration without topology adaptations. We found small differences between the optimizations due to different tuning parameters such as monitoring intervals.

6. RELATED WORK

TARL follows the MAPE-K [13] approach, separating monitoring (topology providers), analysis and planning (TARL rules), and adaptation execution (topology providers). We notice that most adaptation frameworks and rule languages are driven by architecture description languages [15] and do not consider network topology adaptations. Stitch [5], for example, uses an architecture-based self-adaptation approach and operates on components and connectors. TARL could complement adaptation frameworks, such as Rainbow [10] or StarMX [3], to extend their architecture perspective with topologies. Furthermore,

Existing rule-based adaptation concepts, such as rules [1], condition-action rules [6], priority rules [7], policies [2], action policies [14] and Fossa ECA rules [8] are versatile and powerful, but lack expressiveness for efficient topology adaptation rules. *Tactics* in Stitch, for example, use conditions with first order logic expressions, which is insufficient and complex for topology patterns. Their concepts, like offline rule learning [9], rule priorities or QoS-based rule selection, might be useful future extensions for TARL. TARL is the first language tailored for topology adaptations.

A multitude of powerful graph rewriting tools exist, which typically either interpret the rules at runtime (e.g., Viatra 3 [4]) or generate platform-specific code from them (e.g., GrGen.NET [11] for .NET or eMoflon [17] for Java). These (general purpose) tools may be used to evaluate the pattern matching in TARL. In contrast to the concise specification of join and filtering of match sets in TARL, the user has to provide custom implementations for such operations when using general purpose graph rewriting tools directly.

7. CONCLUSION

In this paper, we propose a general model for topology adaptations in networking applications. This model allows the specification of the required topological information to be monitored, the adaptation logic and the conducted adaptation operations. We present TARL, a topology adaptation rule language reflecting this model. Our evaluation shows that the language can express 13 out of 14 considered topology adaptation algorithms from two different application domains. We further developed a runtime environment and ported an existing video streaming application to our approach, which demonstrates the feasibility of the approach.

TARL is an important step toward a general methodology for the development of topology adaptations in networking applications. The abstraction of TARL will enable more sophisticated topology adaptations. Additionally, TARL will allow for automatic reasoning on rules, optimizations and transformations, distributed condition evaluation, adaptation of the required monitoring, and runtime adaptation of the adaptation logic.

Acknowledgments.

This work has been funded by the German Research Foundation (DFG) as part of projects A01 and A02 within the Collaborative Research Center (CRC) 1053 – MAKI. We thank Julius Rückert for his support regarding the integration of TRANSIT with our framework.

8. REFERENCES

[1] J. Adamczyk, R. Chojnacki, M. Jarząb, and K. Zieliński. Rule Engine Based Lightweight Framework for Adaptive and Autonomic Computing. In *Computational Science – ICCS 2008*, volume 5101 of *LNCS*, pages 355–364. 2008.

[2] R. Anthony. A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems. In *IEEE ICAC*, pages 265–276, 2006.

[3] R. Asadollahi, M. Salehie, and L. Tahvildari. StarMX: A framework for developing self-managing Java-based systems. In *SEAMS*, pages 58–67, 2009.

[4] G. Bergmann, I. Dávid, b. Hegedüs et al. Viatra 3: A Reactive Model Transformation Platform. In *Theory and Practice of Model Transformations*, volume 9152 of *LNCS*, pages 101–110. 2015.

[5] S.-W. Cheng and D. Garlan. Stitch: A Language for Architecture-based Self-adaptation. *J. Syst. Softw.*, 85(12):2860–2875, 2012.

[6] F. Fleurey, V. Dehlen, N. Bencomo et al. Modeling and validating dynamic adaptation. In *Models in Software Engineering*, volume 5421 of *LNCS*, pages 97–108. 2009.

[7] F. Fleurey and A. Solberg. A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In *Model Driven Engineering Languages and Systems*, volume 5795 of *LNCS*. 2009.

[8] A. Frömmgen, R. Rehner, M. Lehn, and A. Buchmann. Fossa: Learning ECA Rules for Adaptive Distributed Systems. In *IEEE ICAC*, pages 207–210, 2015.

[9] A. Frömmgen, R. Rehner, M. Lehn, and A. Buchmann. Fossa: Using Genetic Programming to Learn ECA Rules for Adaptive Networking Applications. In *IEEE LCN*, pages 197–200, 2015.

[10] D. Garlan, S.-W. Cheng, A.-C. Huang et al. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10):46–54, 2004.

[11] R. Geiß and M. Kroll. GrGen.NET: A Fast, Expressive, and General Purpose Graph Rewrite Tool. In *Applications of Graph Transformations with Industrial Relevance*, volume 5088 of *LNCS*, pages 568–569. 2008.

[12] B. Karp and H. T. Kung. GPSR: Greedy Perimeter Stateless Routing for Wireless Networks. In *MobiCom*, pages 243–254, 2000.

[13] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[14] J. O. Kephart and R. Das. Achieving Self-Management via Utility Functions. *IEEE Internet Comput.*, 11(1):40–48, 2007.

[15] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. In *FOSE*, pages 259–268, 2007.

[16] L. Krumov, I. Schweizer, D. Bradler, and T. Strufe. Leveraging Network Motifs for the Adaptation of Structured Peer-to-Peer-Networks. In *IEEE GLOBECOM*, pages 1–5, 2010.

[17] E. Leblebici, A. Anjorin, and A. Schürr. Developing emoflon with emoflon. In *Theory and Practice of Model Transformations*, volume 8568 of *LNCS*, pages 138–145. 2014.

[18] M. Lehn, R. Rehner, and A. Buchmann. Distributed Optimization of Event Dissemination Exploiting Interest Clustering. In *IEEE LCN*, pages 328–331, 2013.

[19] N. Li, J. Hou, and L. Sha. Design and Analysis of an MST-Based Topology Control Algorithm. *IEEE Trans. Wireless Commun.*, 4(3):1195–1206, 2005.

[20] X. Liu, L. Xiao, A. Kreling, and Y. Liu. Optimizing Overlay Topology by Reducing Cut Vertices. In *ACM NOSSDAV*, 2006.

[21] A. H. Payberah, J. Dowling, F. Rahimain, and S. Haridi. Distributed optimization of P2P live streaming overlays. *Computing*, 94(8-10):621–647, 2012.

[22] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *ACM/IEEE IPSN*, pages 364–369, 2005.

[23] B. Richerzhagen, D. Stingl, J. Rückert, and R. Steinmetz. Simonstrator: Simulation and Prototyping Platform for Distributed Mobile Applications. In *EAI SIMUtools*, pages 99–108, 2015.

[24] J. Rückert, B. Richerzhagen, E. Lidanski et al. TopT: Supporting Flash Crowd Events in Hybrid Overlay-based Live Streaming. In *IFIP Networking*, pages 1–9, 2015.

[25] I. Schweizer, M. Wagner, D. Bradler et al. kTC - Robust and Adaptive Wireless Ad-hoc Topology Control. In *ICCCN*, pages 1–9, 2012.

[26] H. Shen, Z. Li, and J. Li. A DHT-Aided Chunk-Driven Overlay for Scalable and Efficient Peer-to-Peer Live Streaming. *IEEE Trans. Parallel Distrib. Syst.*, 24(11):2125–2137, 2013.

[27] M. Stein, G. Kulcsár, I. Schweizer et al. Topology Control with Application Constraints. In *IEEE LCN*, pages 229–232, 2015.

[28] J. Suomela. Survey of Local Algorithms. *ACM CSUR*, 45(2):24:1–24:40, 2013.

[29] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *IEEE ICNP*, pages 2–11, 2006.

[30] F. Wang, Y. Xiong, and J. Liu. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *IEEE ICDCS*, pages 1–8, 2007.

[31] Y. Wang. Topology Control for Wireless Sensor Networks. In *Wireless Sensor Networks and Applications*, Signals and Communication Technology, pages 113–147. 2008.

[32] M. Wichtlhuber, B. Richerzhagen, J. Rückert, and D. Hausheer. TRANSIT: Supporting Transitions in Peer-to-Peer Live Video Streaming. In *IFIP Networking*, pages 1–9, 2014.

[33] A. Yao. On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

[34] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.