Application Framework for Programmable Network Control

[1,2]Rudolf Strijkers, [2]Mihai Cristea, [2]Cees de Laat, [1,2]Robert Meijer

[1]TNO Information and Communication Technology, Groningen, The Netherlands,
[2]University of Amsterdam, Amsterdam, The Netherlands
{strijkers, m.l.cristea, delaat}@uva.nl, robert.meijer@tno.nl

Abstract. We present a framework that enables application developers to create complex and application specific network services. The essence of our approach is to utilize programmable network elements to create a software representation of network elements in the application. We show that the typical pattern of an application specific network service is a control loop in which topology, paths, and services are continuously monitored and adjusted to match application specific qualities. We present a platform in which network control applications can be developed and illustrate possible use cases. Based on these use cases, new research questions are identified.

Key words: Distributed Computing, Network Management, Programmable Networks.

1 Introduction

Almost every type of network implements measures to guard against unexpected environmental changes, such as the effects of failing links, changing traffic patterns or the failure of network nodes themselves. Such measures can be considered as optimization of network resources with respect to network robustness. At the basis of the optimization of network resources are programs that control the response of the network to changes in and outside of the network. Moreover, actively controlling network resources is crucial to maintain the network service that is delivered to applications.

Optimizations have a certain penalty in realistic situations. For example, in sensor networks [1] minimizing the transmission power of sensor antennae optimizes battery lifetime, but impacts connectivity. Depending on the application and the actual situation, engineers will choose an optimum. Generally, the optimum network service is application-specific, yet in most networks, application programmers have no control over the network. One reason is that a general applicable, conceptual and technical framework to program the network is absent [2].

In the absence of any notion of specific application demands, as is usually the case, network providers offer typically a best or constant effort network service. Theoretically at least, computer programs can be so specific in their service

requirement and optimal response to disturbances that network providers cannot configure and control the network for such applications anymore. If cloud infrastructures would only run on wind energy, for example, the amount and direction of wind will continuously change the energy available for computing and network resources. In such cases, (partial) control over the network must also be transferred to a computer program, i.e. the application domain, to automate continuous reconfiguration of the infrastructure.

Traditionally, networks have been designed according to well-defined requirements. One could say that at this point application domain knowledge enters the network domain. Conversely, application engineers may use the interface of a given network service, e.g. sockets in the Internet, to include the network in the application logic. Here, we extend the latter approach; any application-specific property of a network service becomes a network control issue programmed in the application domain, i.e. a dynamic user network interface. Moreover, we define the basic framework needed to design and build network control programs in the application domain.

In Section 2 we review state of the art of related areas in programmable networks, overlay network and sensor networks that allow network control from the application domain. Then, in Section 3, the application framework is presented and its functional components are described in Section 4. In Section 5, the implementation and test bed is introduced and Section 6 follows with examples of applications that control networks. The paper ends with conclusions and future work in Section 7.

2 Related Work

A basic approach to develop a programmable network is to use general-purpose computers as Network Elements (NE) and implement C programs that manipulate packet streams and network links [3-5]. The programmable and active network [6, 7] community developed the architectures for dynamic deployment and extensibility of functions in network elements. Other efforts provide programmability in the control plane of networks, while remaining backwards compatible with current Internet technologies [8-11]. These technologies enable network operators to offer better services to applications.

Basically, there are two types of limitations in networks that motivate application control: (1) limited network functions or (2) limited network resources. If the network does not offer enough functionality, a well-known approach is to implement the network functions as part of the application, i.e. create and manipulate a virtualized network (overlay network). If the network has limited resources to accommodate application demands in a best-effort manner, frameworks exist to manage the quality of service on behalf of the application [12-14]. Next, we illustrate some approaches from related network research areas that deal with these limitations.

Overlay networks enable developers to redesign and implement, amongst others, addressing, routing and multicast services optimal to their application domain [15]. Overlay networks are widely used to support specific services, such as distributed hash tables [16], anonymity [17], and message passing [18]. Overlay networks might

lead to sub-optimal utilization of network resources, because the mapping to the physical network resources is not open to the application developer. Moreover, overlay networks essentially duplicate functions offered by the physical network. Recently, some efforts [19] propose to expose physical network properties to applications to improve their mapping to the physical network. Assuming that networks are properly dimensioned, at least from the user's perspective, overlay networks are a straightforward solution to support their specific network service requirements.

Sensor networks illustrate best limitations in network resources. Sensor networks motivate tight integration of applications and network services [20]. Because of the resource constraints, sensor network designers attempt to use the scarce resources efficiently and various approaches to program sensor networks have been developed [21]. In *macroprogramming* [22], high-level programs use an intermediate language to abstract away concurrency and communication aspects in sensor application programming. A compiler translates the programs into basic instructions for individual nodes, and takes communication characteristics into account. In TinyDB [23], communication is integrated with a data query mechanism. *Macroprogramming* and TinyDB show that with a framework that structures the design space of network control applications, it becomes possible to design and implement reusable components for new applications.

Our research in advanced applications of networks [24-30] shows that applications have different optimal network services. Existing network management systems do offer APIs to configure network services [31]. Such APIs implement the network abstractions chosen by the network operator. We found that our use cases in hybrid networks and sensor networks require more flexible and specific network services than those designed and implemented by network operators. Because the application domain offers developers more flexibility, it might be more practical to implement network services as part of the application. Hence, we developed a model that enables developers to program networks as part of their application [32]. The resulting framework, User Programmable Virtualized Networks (UPVN), models the *interworking* between networks and applications and provides a conceptual framework to investigate design patterns of application-specific network services. Here, we shortly introduce the model.

In UPVN (Figure 1), individual NEs are regarded as resources, which are used directly or through the Internet (open lines) as components in application programs. A NE component (NC) can be seen as a manifestation of the NE in the application, i.e. a virtualized NE. Consequently, all virtualized NEs together create a virtualized network, allowing interaction with user programs. To accommodate application specific packet processing, to set particular parameters of the NE, and to facilitate other functions NEs play in a UPVN, NEs have the ability to deploy Application Components (ACs).

UPVN's development is application driven; creating only those facilities that are crucial for applications while other operations remain automated. The NE uses technologies, such as Grid- and web services, to expose interfaces on the Internet. Through the interfaces a NE exposes, various applications interact simultaneously

with the NE. As such, each application is capable to optimize the behavior of the NE accordingly. During application development, the NE appears as a software object, i.e. Network Component (NC), in the development environment. During run-time, state of the art technology allows dynamic extension of the set of NEs the applications interacts with.

The UPVN model leads to a practical framework in which network control is implemented as part of application domain programs and in which network services and optimizations are expressed in user-definable qualities. In the past, we developed a prototype UPVN that showed that the approach is feasible [33]. In Section 4 and 5, we present the design and implementation of a prototype that includes the control concepts we propose. In the following section, the application framework for programmable network control is introduced.

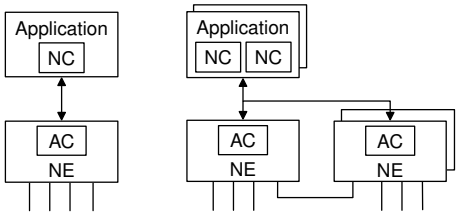

Fig. 1. Interworking model of applications and networks.

3 Application Framework for Network Control

Programmable network element technologies support dynamic network service composition for applications that need new network functions, such as network embedded trans coding of video streams. If changes occur in the network, however, applications must adapt to the new situation. The adaptation process may be at the end-points, such as in TCP flow control process but may also be in the network, such as a process that changes the edge weights of a shortest path routing protocol [34]. The adaptation process typically consists of (1) inferring (possibly incomplete) network information, (2) calculating network state (3) and adjusting the network to a configuration that leads towards the desired optimum. A closed-loop control model, a well-known model in control theory to influence the behavior of a dynamic system [35], provides a minimal framework for network control (Figure 2).

In order to match the network to a state that is optimal to an application, the application has to collect (possibly incomplete) network information. The application developer chooses application specific abstractions (NC_x) to update a model the application uses internally. The application combines state information from all or a subset of NEs to update the internal model. In principle, the internal model can also include non-network related information, such as computing or hosting costs, sensor information and service level agreements.

The control application applies an algorithm to find the actions (NC_y) needed to adjust the network behavior in such a way that it matches the application needs (e.g. a

stable, optimized state), which are described by the reference. To implement changes in the network, the control application translates decisions into instructions, such as create, forward or drop packets specific to each NE involved in the application. This means that the system needs to provide a distributed transaction monitor to keep network manipulations that involve multiple NE consistent.

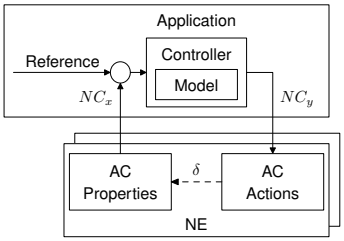

Fig. 2. The application framework to control networks contains a control loop.

In control theory, a measurement (AC Properties) from the system is subtracted from a reference value, which leads to an error value as input for the control application. In our framework, the measurements (AC Properties) that represent network state may use different metrics compared to the controlled state (AC Actions). For example, a controller may manipulate edge weights in shortest path routing based on throughput information. Such a scenario is meaningful if the relation between throughput and edge weights (δ) is known or can be learnt and would be useful to dynamically distribute traffic to avoid congestion, for example [34].

Applications exchange information ($NC_{x,y}$) with NEs over a communication network, possibly over the same network the application is controlling (in-band). Even though application developers may have access to a separate management network, the communication path between network and application complicates the design and validation of the controller. Network properties, such as latency and packet loss, limit the amount of information that can be exchanged or synchronized. So, NE state information can become incomplete, inaccurate or aged. The application developer has to understand the limits in information exchange of a given network, i.e. observability, when designing the control application.

This section introduced the abstractions needed to provide the basic framework for network control in the application domain. Next, the details related to interworking of applications and networks that lead to a functional model are described.

4 Functional Components

The OSI reference model organizes the interworking of applications and networks in seven layers [36]. The design principle of layering allows decomposition of a complex problem, but application specific details may be lost in the process. If network elements are virtualized in software, the application interface to the software (NCs) can be fine-tuned to the specific problem domain. However, the fine-tuning

might lead to an application specific organization of network functions. Here, we define the organization of functional components to support fine-tuning of the application interface and organization of network functions. The functional organization preserves the context of the NEs by creating and managing the software representation of NEs in the application domain. For example, an application can use the software representation of NEs to manipulate traffic of a single strategic point in the network for filtering or anomaly detection purposes.

We identify three layers of abstraction in a distributed program: network element execution environment, middleware/orchestration, and application code. The latter can be subdivided in two sub layers, namely the programming environment providing reusable components such as programming libraries, and the application program. The result is a four-layer architecture (Figure 3). Clearly, the architecture resulting from the application point of view is similar to programmable network architectures [6]. However, the functional components between the application and programmable network need to be further defined to support network control from the application domain and is described next.

The orchestration layer (2) facilitates the interworking of software objects and ACs located on individual NEs (1). The orchestration layer may also supports basic mechanisms, such as discovery services, brokers, billing services, authorization, etc. The usefulness of these services depends on the network environment and application. In sensor networks, for example, there just may not be enough computational and storage resources to support an elaborate set of services.

The programming environment, layer (3), provides the NC implementation and reusable components, such as a Distributed Transaction Monitor (DTM) or breadth-first search algorithm, to support programming of a collection of NCs. Depending on the network environment, some abstractions can be implemented in the ACs, as a library in the programming environment or both. For example, the application developer might want to program network element interactions in a non-blocking manner. Hence, either the programming environment or the orchestration layer must facilitate non-blocking interaction mechanisms between ACs and NCs. In our implementation (Section 3) we use message passing in the orchestration layer and implemented (an easier to program) blocking interface to the application (Section 5).

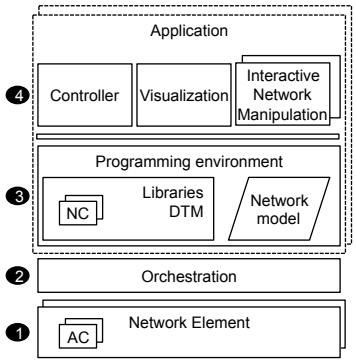

Fig. 3. Four functional layers characterize practical application domain network control.

Because network control is now part of the application domain (layer 4), developers can benefit from a large amount of existing software to implement network control programs. A characteristic of the control applications is that they operate on data structures that represent the network state. Therefore, the programming environment (3) explicitly contains a model of the network and the orchestration layer must supply the data with which the model can be updated. In Section 6, we discuss issues related to the accuracy of the network model.

Some applications support the construction of a network model that is close to mathematical concepts, such as graphs. The Mathematica [37] environment, for example, contains a graph data structure, which can be used as a basis for control applications that require graph algorithms. By enabling dynamic updates of network state into the Mathematica graph data structure, domain experts can simply apply graph algorithms to find and remove (through network manipulation) articulation vertices; vertices that may disconnect a graph. Besides control, the application layer can also include visualization or other means of interaction with the network. The integration with toolboxes, such as those available in Mathematica, makes the application layer a powerful environment to develop network control applications.

5 Implementation and Test Bed

In the preceding sections, we introduced the framework for control applications as well as a four-layered functional model to implement such applications. We developed a test bed according to the presented functional model (Figure 3) to gain practical insight in the implementation of the application framework to support network control programs. The test bed implements the first three functional layers and enables further exploration of the network control applications that are part of the fourth layer.

5.1 Hardware

The test bed consists of eight machines (four dual processor AMD Opteron with 16GB RAM and dual port 10Gb NICs and four Sun Fire X4100 with 4GB RAM and 1Gb NICs) interconnected by two 1Gb switches and a Dell hybrid 1/10Gb switch. All machines run VMWare [38] ESXi hypervisor software and the virtual hardware is centrally managed and monitored with VMware vSphere management software. The test bed was bootstrapped with one Linux instance containing the software we developed, and iteratively grown to 20 instances to create a non-trivial configuration of networks and computers (Figure 4).

The setup involves two datacenter locations: a virtual infrastructure running in our datacenter in Groningen and an interactive programming environment including an interface to a multi-touch table running in our lab in Amsterdam. The multi-touch table enables users to interact with NCs (Section 6). The two locations are connected by two OSI-Layer 2 Virtual Private Networks (VPN) on basis of OpenVPN [39]: one for control traffic and one for data traffic. At the receiving host in Amsterdam, the

control and data networks are separated by VLANs.

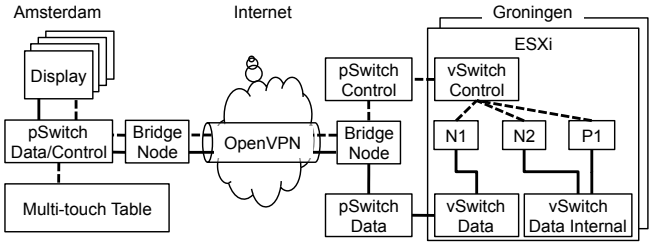

Fig. 4. Test bed and network connectivity.

5.2 Software

The primary purpose of developing a prototype is to gain insight in the challenges and details to control a network from applications that require dynamic traffic manipulation, and to enable experiments with various network control mechanisms. The implementation combines several open source software tools into one NE platform. We provide a global overview of the software that implements the functional layers.

```
(netfilter_fetch_in) >(fpl_tbs,expression="TOKEN") \
  >(fpl_ipdest,expression="DST_IP") >(skb_transmit)
```

Fig. 5. A Streamline request in which packets are taken from the Linux Netfilter [41] hook, then filtered by token and the IP destination overwritten.

Packet Processing and Token Networking. Fine-grained packet processing and manipulation facilities are implemented in Streamline [4], a tool originally developed for high-speed packet filtering and similar to other approaches presented in literature (Section 2). However, Streamline differs from other approaches by providing a simple and flexible query language to manipulate filter graphs on the fly (Figure 5) and a packet processing language FPL [40]. In addition, Streamline also allows dynamic loading of kernel modules that provide specific packet manipulation functions.

We extended Streamline to support insertion, removal and filtering of tags in the IPv4 options field, which allows us to bind ACs to network traffic. A Streamline expression defines a chain of packet processing modules, which describes the network behavior for a particular application on a NE. Filters, such as *fpl_tbs* allow packets with specific tags to pass through a specific chain of packet processing modules. The expression is calculated for each NE separately by the control software and a distributed transaction monitor manages loading of each expression on the subsequent nodes to provision a path, for example.

Orchestration Software. The orchestration of ACs in the programmable network is implemented in Java. ACs available to applications, such as Streamline, are wrapped

by Java objects. Network elements communicate using a peer-to-peer model. ACs register as a service on the network element. Each network element knows at least one peer to which it can connect (the controller). Currently, all peers connect to a single known controller, which provide basic message-passing functions over TCP sockets. The controller also provides basic services that involve more than one NE, such as a distributed transaction monitor or topology discovery. The basic services are implemented as a set of ACs and can be used by network control applications. Currently only a single instance of the controller is used. Creating more controllers on-demand is a topic for future investigation. (Section 6).

Network Model. Our implementation provides various active and passive monitoring ACs that enables network control applications to create and maintain a network model:

Ping provides basic information about latency and jitter,

Network Mapper (NMap) [42] can detect nodes in the broadcast domain of an interface with ARP, and

/proc/dev/net is used to retrieve basic throughput information from the Linux kernel,

Uptime collects CPU load information.

The controller contains a Dispatcher AC that allows other ACs to subscribe to events, such as NEs registering to or detaching from the network and is the entry point for peers that connect to the control network. The Dispatcher AC subscribes to all known network elements and triggers network discovery requests when a new NE registers, consequently updating its network model to the new network state.

AC Management. Management functions, such as starting, stopping and manipulating AC of the programmable network implementation, are implemented in

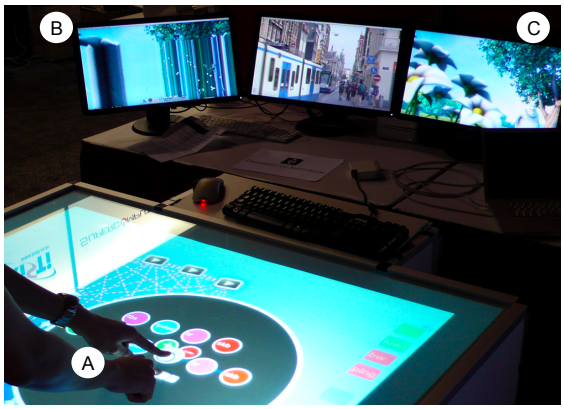

Fig. 6. A multi-touch table enables direct manipulation of programmable network components of 20 virtual machines. A user (a) modifies a sampler component of a streamline graph that multicasts a video to screen (b) and (c). As a result, the stream of (b) is distorted, while the other remains normal.

the Ruby [43] programming language. This allows new network behavior to be added at runtime, e.g. Java classes, kernel modules or installation of complete applications.

For example, a ruby script with instructions to compile new code for Streamline and insert it into the kernel can be remotely executed on NEs.

6 Network Control Programs

We showed a practical implementation of the model in Section 4, which enables a straightforward prototyping of network control programs. To test the setup an interface to view and modify the state of NCs was built. It allows manipulation of video streams produced by several nodes, which can be displayed on computers with a screen attached. By manipulating the NCs, a user can interact with the programmable network: create and modify paths and modify NE parameters, such as the packet processing chains of Streamline. We successfully demonstrated the setup at Super Computing 2008 in Austin, TX [44] (Figure 6).

We developed an interactive programming environment with Mathematica, which enabled automation of the possible user manipulations in the setup. Combining Mathematica with programmable networks allows advanced, yet straightforward implementation of network control applications. We implemented a Java adapter between Mathematica and the Management Agent (orchestration layer). The Java adapter deals with limitations of Mathematica's, such as real-time polling of the network, while being responsive to user input at the same time.

The Java adapter enables the Monitor AC to trigger the continuous updates of a number of data structures in the Mathematica kernel, such as *theNetwork* or

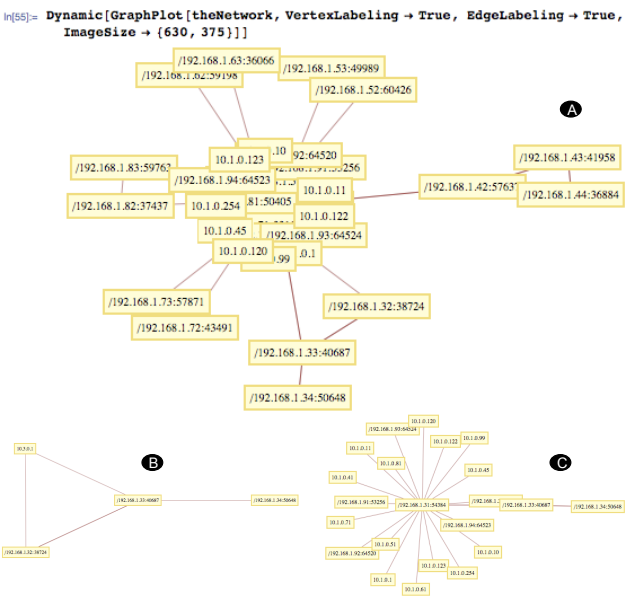

Fig. 7. Mathematica's function *Dynamic[]* facilitates continuous reevaluation of network state. The statement redraws the graph every time *theNetwork* data structure is updated with information of the network (a). Picture (b) and (c) show two stages of topology discovery.

thptNetwork, and facilitates the development of control applications in Mathematica. An elementary control application is one that visualizes the network state while the data structures are updated (Figure 7). For example, the current IP network topology can be displayed while the discovery of the network is in progress; fully discovered in (a) and two intermediate steps (b) and (c). Another visualization example maps throughput measurements on a 3D contour plot (Figure 8). We also implemented various control applications using the test bed. For example, two control applications avoiding congestion were implemented by switching paths and by dropping packets on basis of throughput measurements. Another control application was to developed to continuously find and provision disjoint shortest paths [26]. Based on the experiments, we identify new research questions.

Application developers have to consider the accuracy of the network model. For network properties as throughput and delay some range of error can be tolerated. However, applications that require exact shortest paths require accurate topology information. The accuracy of the network model is influenced by the rate at which state information is (1) generated, (2) transported and (3) processed. At least (2) and (3) have architectural consequences for the control loop. One possible architectural consequence is to divide the network in multiple separately controlled domains, similar to areas in OSPF. In one extreme, dividing up the network into smaller individual control domains eventually leads to a fully decentralized architecture, i.e. peer to peer networks. In the other extreme, if network state can be generated, transported to and processed fast enough by one controller, then for practical purposes a centralized implementation might be preferred.

Application developers have to make a trade-off between state exchange and the processing capabilities of network elements. For example, an application that finds and removes articulation vertices can run as (1) a centralized component or, in the

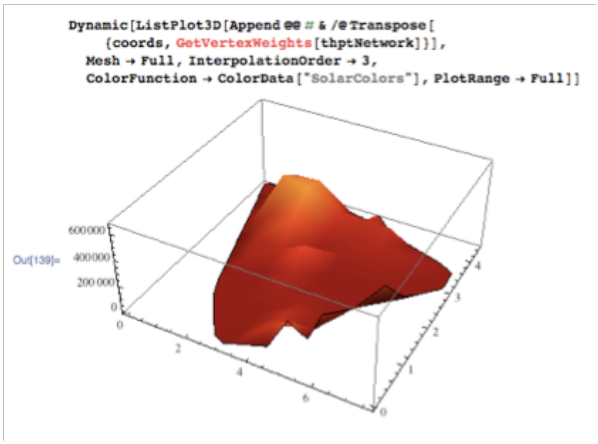

Fig. 8. Network throughput of the test bed visualized in Mathematica. The vertex weights in the *thptNetwork* data structure are updated with throughput values from the programmable network. The network topology is mapped to the x-y plane and throughput to the z-axis (in bytes per second). This way, a user can detect busy spots network and write algorithms to avoid such spots.

other extreme, (2) can run on each NE under its control. Because the computation of articulation vertices requires full topology knowledge, running the application on each NE (2) requires additional mechanisms to update and synchronizes changes in topology. Between centralized and decentralized implementations of control loops many architectural variants exist. Likewise, an enormous variety of control algorithms can be expected. On these points applications programmers would benefit from research [45] on design patterns of control loops.

7 Conclusion and Future Work

Until now, engineers optimize networks at design time and independent of application engineers. Examples from sensor networks, hybrid networks and overlay networks show a need to control networks at run-time. Past efforts created the programmable network element technologies to support dynamic network service composition. In this paper, we use these technologies in a framework for network service development in which each programmable network element has a software representation in a possibly distributed application. We presented an implementation of the framework and several network control applications.

Our implementations are limited to a single application that controls the network. In case many applications want control over the network, another control application is needed to manage (conflicting) resource demands, i.e. an operating system for networks. In the future, however, it can be expected that network management systems support mechanisms to host and run applications on the network. Recent research also continues in this direction (Section 2). More experience is needed to create reusable software components that enable and simplify control application development for large networks.

Control loops are a fundamental part of applications that optimize a specific network service as a response to changes in or outside the network. In subsequent research we shall determine the operational properties of a control application (e.g. how accurate is a given network state, what is the delay between network events and the application's ability to react, how fast can failures be detected). We have shown that architectural consequences can be expected when changes in the network occur faster than a single control loop can effectuate new adjustments, e.g. in large or unstable networks. In this case, the application framework needs to support decentralized network control. Hence, to extend the application framework to support multi-domain, multi-scale network control is a topic for further research.

Acknowledgments

We thank Wolfgang Mühlbauer, Burkhard Stiller and Bernhard Plattner for their comments and support.

References

1. Culler, D., Estrin, D., Srivastava, M.: Guest Editors' Introduction: Overview of Sensor Networks. IEEE Computer 37 (2004) 41-49
2. Ng, T.S.E., Yan, H.: Towards a framework for network control composition. Proceedings of the 2006 SIGCOMM workshop on Internet network management. ACM, Pisa, Italy (2006)
3. Elischer, J., Cobbs, A.: FreeBSD Netgraph pluggable network stack http://www.freebsd.org/, accessed at 10 August 2009
4. Bos, H., Bruijn, W.d., Cristea, M., Nguyen, T., Portokalidis, G.: FFPF: Fairly Fast Packet Filters. OSDI (2004)
5. Morris, R., Kohler, E., Jannotti, J., Kaashoek, M.F.: The Click modular router. SIGOPS Oper. Syst. Rev. 33 (1999) 217-231
6. Campbell, A.T., Meer, H.G.D., Kounavis, M.E., Miki, K., Vicente, J.B., Villela, D.: A survey of programmable networks. SIGCOMM Comput. Commun. Rev. 29 (1999) 7-23
7. Tennenhouse, D.L., Wetherall, D.J.: Towards an active network architecture. SIGCOMM Comput. Commun. Rev. 37 (2007) 81-94
8. Wang, W.M., Dong, L.G., Bin, Z.G.: Analysis and implementation of an open programmable router based on forwarding and control element separation. Journal of Computer Science and Technology 23 (2008) 769-779
9. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38 (2008) 69-74
10. Casado, M., Freedman, M.J., Pettit, J., Luo, J., Gude, N., McKeown, N., Shenker, S.: Rethinking Enterprise Network Control. Networking, IEEE/ACM Transactions on 17 (2009) 1270-1283
11. Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S.: NOX: towards an operating system for networks. SIGCOMM Comput. Commun. Rev. 38 (2008)
12. Braden, R., Clark, D., Shenker, S.: Integrated Services in the Internet Architecture: an Overview. RFC1633 (1994)
13. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. RFC2475 (1998)
14. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. RFC3031 (2001)
15. Lua, K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. Communications Surveys & Tutorials, IEEE (2005) 72-93
16. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, San Diego, California, United States (2001)
17. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The Second-Generation Onion Router. 13th USENIX Security Symposium (2004) 303-320
18. Open MPI: Open Source High Performance Computing http://www.open-mpi.org/, accessed at 11 August 2009
19. Xie, H., Yang, Y.R., Krishnamurthy, A., Liu, Y.G., Silberschatz, A.: P4p: provider portal for applications. SIGCOMM Comput. Commun. Rev. 38 (2008) ACM--362
20. Romer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wireless Communications 11 (2004) 54-61
21. Royer, E.M., Chai-Keong, T.: A review of current routing protocols for ad hoc mobile wireless networks. Personal Communications, IEEE 6 (1999) 46-55

22. Newton, R., Arvind, R., Welsh, M.: Building up to macroprogramming: an intermediate language for sensor networks. Proceedings of the 4th international symposium on Information processing in sensor networks. IEEE Press, Los Angeles, California (2005)
23. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. 30 (2005) 122-173
24. Meijer, R.J., Koelewijn, A.R.: The Development of an Early Warning System for Dike Failures. 1st International Conference and Exhibition on WATERSIDE SECURITY, Copenhagen, Denmark (2008)
25. Cristea, M., Strijkers, R.J., Marchal, D., Gommans, L., Laat, C.d., Meijer, R.J.: Supporting Communities in Programmable Networks: gTBN. IFIP Integrated Management 2009, New York (2009)
26. Strijkers, R.J., Meijer, R.J.: Integrating networks with Mathematica. 9th International Mathematica Symposium 2008, Maastricht (2008)
27. Cook, G.: ICT and E-Science as an Innovation Platform in The Netherlands. Cook Report on Internet Protocol. Cook Network Consultants (2009)
28. Portegies Zwart, S., Ishiyama, T., Groen, D., Nitadori, K., Makino, J., Laat, C.d., McMillan, S., Hiraki, K., Harfst, S., Grosso, P.: Simulating the universe on an intercontinental grid of supercomputers. Submitted to IEEE Computer (2009)
29. Kruithof, N., Marchal, D.: Real-time Software Correlation. INGRID Workshop (2008)
30. Strijkers, R., Cristea, M., Khorkov, V., Marchal, D., Belloum, A., Laat, C.d., Meijer, R.: Network Resource Control for Grid Workflow Management Systems. SWF2010. IEEE, Miami, Florida (2010)
31. Haggerty, P., Seetharaman, K.: The benefits of CORBA-based network management. Commun. ACM 41 (1998) 73-79
32. Meijer, R.J., Strijkers, R.J., Gommans, L., de Laat, C.: User Programmable Virtualized Networks. Proceedings of IEEE International Conference on e-Science and Grid Computing. IEEE Computer Society (2006)
33. Strijkers, R.J.: The Network is in the Computer. Master Thesis. Informatics Institute, University of Amsterdam, Amsterdam (2009)
34. Fortz, B., Rexford, J., Thorup, M.: Traffic engineering with traditional IP routing protocols. Communications Magazine, IEEE 40 (2002) 118-124
35. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
36. Zimmermann, H.: OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection. Communications, IEEE Transactions on 28 (1980) 425-432
37. Wolfram Mathematica http://www.wolfram.com/mathematica/, accessed at 2 August 2007
38. VMWare http://www.vmware.com, accessed at 2 August 2007
39. OpenVPN http://www.openvpn.net/, accessed at 14 August 2009
40. Cristea, M., de Bruijn, W., Bos, H.: FPL-3: towards language support for distributed packet processing. Proceedings of IFIP Networking '05 (2005)
41. Linux Netfilter http://www.netfilter.org, accessed at 17 August 2009
42. Network Mapper http://nmap.org, accessed at 7 April 2008
43. Thomas, D., Fowler, C., Hunt, A.: Programming Ruby. Pragmatic Bookshelf (2004)
44. Strijkers, R., Muller, L., Cristea, M., Belleman, R., Laat, C.d., Sloot, P., Meijer, R.: Interactive Control over a Programmable Computer Network using a Multi-touch Surface. ICCS 2009. LNCS, Baton Rouge, Louisiana (2009)
45. Feitelson, D.G.: Distributed Hierarchical Control for Parallel Processing. Computer 23 (1990) 65-77