

User Programmable Virtualized Networks

^{1,2}Robert J. Meijer, ¹Rudolf J. Strijkers, ¹Leon Gommans, ¹Cees de Laat

¹University of Amsterdam, Amsterdam, The Netherlands,

²TNO ICT, Delft, The Netherlands

rmeijer@science.uva.nl, rjstrijk@science.uva.nl, lgommans@science.uva.nl, delaat@science.uva.nl

Abstract

This paper introduces the concept of a User Programmable Virtualized Network, which allows networks to deliver application specific services using network element components that developers can program as part of a users application. The use of special tokens in data or control packets is the basis of a practical, yet powerful security and AAA framework. This framework allows for implementations with a low footprint that can operate in a multi domain network operator environment. We demonstrate the ease with which one can build applications and address networking problems as they appear for example in sensor networks.

1. Introduction

The concepts defined in the OSI model for the interaction between networks, end systems and their applications, are widely accepted [1]. International telecommunication infrastructures and the Internet are based on these concepts. Because details like network topology are irrelevant to most applications, OSI considers only end-to-end transport services.

What if network providers cannot understand all of your network service demands anymore? What if the network cannot be over-provisioned due to the involved costs? If one detects that an IP router will fail shortly, how can we route a VoIP stream over an alternate path before the router actually fails and before the users notice anything? Do video streams of a burning car have priority over those of collided cars not far away in a heavily congested network? In such cases, there is a need to tune the network service to the demands of the users and their application programs; one has to facilitate application specific networking.

Neither the set of Internet protocols, nor a network management system (NMS) provides practical control interfaces to individual network nodes. The services of

TCP and UDP are often used through socket APIs. Socket APIs however, hide most of the network details such as topology. In theory, using the NMS would be one way for the application programmer to discover and possibly control network elements, such as Cisco's Active Networks Abstraction [2]. The span of control of an NMS however, is typically restricted to a single network operator domain. Furthermore, NMSs are designed to support operators and not end user programs. Moreover, only operators, not end users, are allowed to use the NMS.

The concept of programmable networks is sufficiently well known to create concepts and technologies that support application specific networking [3]. The concepts differ in how applications interface with network nodes. Basically there are three variants: agents, active messages (also known as active networks) and remote method invocations (RMI) [4]. In short, agents are programs that travel from node to node, active messages are network packets extended with application code and webservices are a great example of RMI. The IETF ForCES working group basically standardizes common elements in IP routers [5]. One of the benefits is that elements may be changed or added and combined with web and Grid services [6][7]. ForCES does not have a security concept that is practical in a multi domain network (see Section 2.3). Currently, years of developments in programmable networks have led to complex frameworks with corresponding complex technologies. This prevented the emergence of killer applications and market impact [8].

Sensor networks are frequently designed to operate in a very dynamic context, in which sudden environmental changes may cause parts of the network to become isolated. This has inspired the ad-hoc networking concept, where a system of identically programmed sensors collectively supports a telecommunication service amongst themselves [9][10]. Research, however, is predominantly focused on topics as autonomy and self-organization of sensor systems [9]. Little, if any, attention is given to the interaction between end-user applications and the sensor network and to the fact that in

realistic situations sensor networks belong to multiple organizations.

2. UPVN

User Programmable Virtualized Networks (UPVNs) is a concept that allows end-user applications to interact closely with network elements (NEs) such as IP routers. Ultimately UPVN concepts should be applied in sensor networks where technologies with small footprints are required. Therefore UPVN's development is application driven; creating only those facilities that are crucial for applications. Yet, UPVN uses much of the concepts presented in [6][7]. Most importantly, using Grid concepts, one can regard individual NEs as resources, which are exploited through the Internet 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, as shown in Figure 1. Hence, our concept is named User Programmable Virtualized Network or, abbreviated, UPVN.

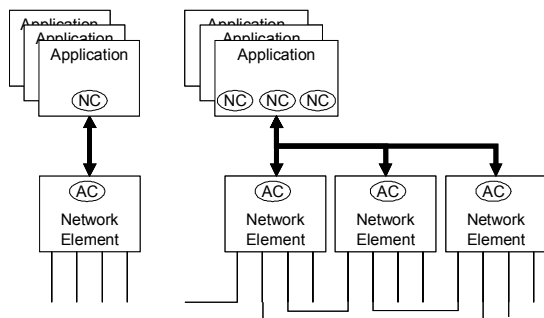


Figure 1. UPVNs model for the interworking of a network element and applications. Left: the virtualization of a network element (NE) as a NE Component (NC). Similarly, specific Application Components (AC) run on a NE. Right: the virtualization of a network. The NEs are either interconnected directly to each or connect to the Internet (open lines).

For telephone conversations, the network access provider is responsible for setting up the end-to-end connection. An Internet service provider is used to connect to the Internet. For application specific networking, this is generally not possible. NEs along a path can belong to different network owners. Technical and other (e.g. financial) conditions to access and use NEs along the path may differ such that the access provider cannot judge if NEs are good enough for the application. The application must make its own judgments. We describe in [11], that in multi domain situa-

tions, the use of tokens in data packets provides a practical and robust security and AAA framework.

A token can be considered as a reference to a service the user has agreed with the network operator. An application can interact with the NC in order to reach this agreement. By subsequently allowing applications to insert these tokens in a secure manner into the data or control stream of a network, applications are able to signal the NEs that it is authorized to use the agreed service and to be accounted for it. Compared to the alternative, the integration of back office systems of network operators, a token based security and AAA mechanism is easy to implement, see Section 2.3.

2.1. Virtualized network element

Figure 1 shows UPVN's interworking model of a NE and a network of NE's with applications. The NE uses technologies, such as Grid- and webservices, 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 the design phase of the application, the NE appears as an object, called a network component (NC), in the development environment. During run time, our model, as well as state of the art technology, allows dynamic extension of the set of NEs the applications interacts with.

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). ForCES [5] and the Click Router [12] deploy similar concepts. In UPVNs, the delivery method of ACs to the NEs is regarded to be application specific. As one of the available methods, one could use the NC to deliver the AC to the NE. In any case, care should be taken to avoid problems caused by ACs acting on the AC to NC network traffic, e.g. by using tokens.

ACs can either operate directly on the packets, or send them to the NC. Then, through the NC, the application can manipulate the packet and send it back. As an interesting consequence applications can also be constructed that transfer the received packets via any of the NCs into another NE. This NE can put the packet via an AC on a network link or into the IP routing engine of the NE. We have called this process "packet warping", as packets disappear in one NE from the network, to reappear in a supposedly disjoint NE somewhere else.

2.2. NEs and the Internet

NEs have IP routing capability and can be connected to other NEs and the Internet. Consider Figure 2, which shows eleven NEs. In UPVN NEs can disappear and reappear, similar to what happens in mobile sensor networks. The Internet is depicted as a network cloud, appearing several times in Figure 2. NEs like 3, 7 and 8 have direct connections to other NEs (1, 4 and 5) via links that are completely under NE control. They may carry Internet traffic, but a NE could exert manipulation and control on it.

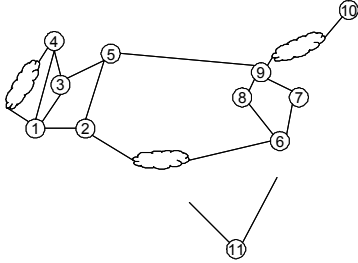


Figure 2. UPVN regards two kinds of links: those completely managed by NEs (straight lines) and Internet links (clouds).

Links such as between 9 and 10, that are not under control of a NE pair, carry in our model regular, best effort Internet traffic. NEs like 1 and 4 are linked also via the Internet. NE 10 is reachable only through the Internet. NE 11 is not connected at all yet, but could be, either directly or through the Internet. A manual or application driven action could be responsible for this.

Another way to look at Figure 2 is that the Internet is encapsulated with NEs. Because of this the Internet is implicitly virtualized in UPVN, allowing applications a specific multipoint interaction with the Internet. UPVNs approach to application specific networking has deep impact on the behavior of NEs. The processing of packets is now, opposite to IP routers, inherently stateful: two identical packets sequentially send to a NE may be processed differently: for example processing may stop if the stored value is depleted in a pre-paid account.

Important questions are: “What functions of the NE should be exposed in the NC?” and “What functions should ACs have?” On basis of our experiences with virtualizations of optical switches [13], we would choose for the ability to intercept, re-route, and reserve resources such as transmission lines. It is however hard to make choices. For some applications, control over the TCP maximum segment size is crucial. Others would like to use a certain transmission path if the price is right at the right moment. In this paper we fo-

cus on important service aspects such as forwarding, transactions, topology and quality.

2.3. Token based NE services

Both the OSI model and the IP define various Transport and Link Layer services. Together they create end-to-end services. Hence packets do not contain fields explicitly designed to trigger the invocation of specific AAA mechanisms and the correct NE services. In [11] we describe a secure token mechanism, which we use in UPVN in cases where NE services or the packets themselves have to be secured. In UPVN, tokens can be added for example to the IP options field.

Tokens can be cryptographic products using a key issued by the application. The result is placed in the IP options field; whilst the key is securely provisioned via a NC to a “Token AC” in the NE. Cryptographic operations of the Token AC validate the token. Logic in the Token AC selects the proper AAA AC and has subsequently other AC services executed. If a packet does not contain tokens, if tokens are not recognized, or if no ACs are configured who operate on regular IP packets, the packet follows the default IP processing in a NE. Operators of different network domains only need to understand the cryptography of the token and its associated key and do not have to integrate each others AAA and trust systems [11]. For this reason, UPVNs use of tokens can ignore the discouragement of the ForCES working group to create access to NEs for end-user applications from other network operator domains. Finally, security and AAA issues related to the use of the NC mechanism could be addressed by Grid and webservice technologies, although in the case of sensor network applications, they may require too many resources.

2.4. Implementation of the NE service

To gain experience, and to develop the UPVN concept toward small footprint implementations, a prototype NE was implemented. This NE is basically a PC with Ethernet cards running the Linux OS. The OS was modified to add functions in kernel and user space. Figure 3 shows the architecture of the NE. The design is straightforward: we have encapsulated the in- and outputs of the Linux Kernel IP routing and forwarding functions with facilities to filter and to inject packets. The packet injection and filter facilities use the `netfilter` function `libnetfilter_queue` (the successor of `libipq`).

`Libnetfilter_queue` acts both in the kernel and user space of the OS. ACs, which execute in user space, can use it to exercise detailed control over the IP

traffic. Packets can be dropped, generated and modified by the AC and, through the AC-NC connection, even by applications. ACs are stored in the Application Component Collection (ACC). By manipulating the ACC, the Application Component Manager (ACM) controls the life time of an AC and the tree like chaining of AC. This enables ACs to operate on one or more copies of packets or packets already modified by ACs.

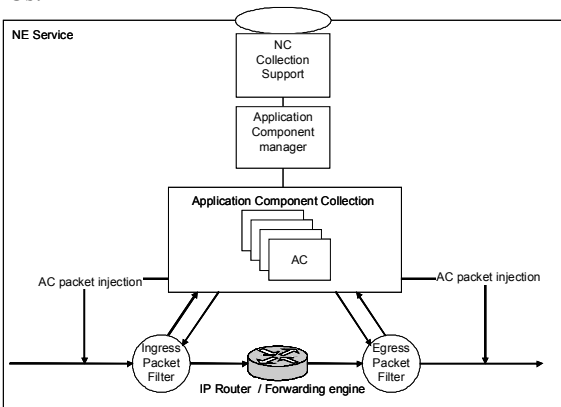


Figure 3. Overview of the NE implementation in Linux.

The NC Collection Support module (NCS) takes care of the NE service interface and contains functions to identify the NE and other functions that are helpful in case the NC becomes part of collections in applications. Amongst others, NCS exposes the AdjNe AC that discovers the neighbors of a NE and identifies them as an NE or regular Internet node (router, end system).

The user-space programs in the NE are coded in Ruby [14], a language allowing an efficient first-time implementation of UPVN concepts. With Ruby's DRb (a remote method invocation facility) the NE services interface was created, avoiding for the moment the complexities and consequences of webservices on a NE: parsers, web servers and performance implications [8]. Furthermore, it allowed exploring network service creation with clear concise Ruby programming. In section 4.1 the use of webservices is discussed.

3. Virtual NEs and virtual links

As stated in section 2.1 in UPVN it is possible to warp packets via the application between NEs. This, and the fact that the NC shields details of the NE, allows the existence of virtual links and virtual NEs, which reside *inside* the application.

Figure 4 shows three virtual network elements (VNEs). These application components may function

such that it appears to have network connections between both to each other and to real NEs via NCs. Such connections are easily implemented using socket APIs and local IP addresses. VNEs can be implemented in any way the application developer seems apt. It is rather straightforward to imagine that VNEs run the same algorithms as NEs. Indeed, VNEs might contain complete IP routing functionality even with assigned IP addresses and as such bring the Internet into the applications.

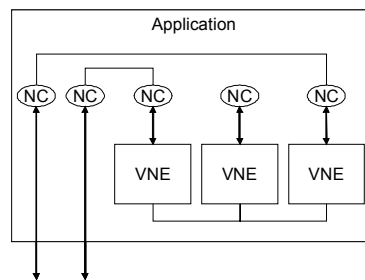


Figure 4. Since the NC hides details of the NE implementation, applications are unable to discriminate between real and virtual NEs (VNE) that are application modules.

The concept of VNEs inspires to even more exotic constructs. The application developer is free to include NCs of NEs and VNEs in a VNE, the start of recursive pattern. A potential very useful application is one in which the application developer pairs a VNE with a real NE. Parameters indicative for the performance of the NE are obtained by the VNE through an incorporated NC. Applications may then interact for certain matters with the VNE. E.g. if the NE reports its links to the VNE, topology inquiries can be done to the VNEs.

4. The services of virtualized networks

An application, which contains one or more NCs, is said to contain a virtualized network. An application program might contain only the NC of single NE even if the network contains hundreds of other NEs from which NCs can be obtained. Regardless if an application contains only one or all NEs, we say the network has a manifestation in the application. There are good reasons for manifestations with a single NC, for instance the filtering of traffic at a strategic point. Therefore, details of manifestations of networks are in UPVN considered application specific, requiring no general frameworks, facilities and structures. UPVN lacks general discovery services, brokers, billing services, AAA servers, etc. The usefulness of this depends on the application. In sensor networks, for instance, there might just be not enough infrastructure around to support an elaborate framework.

The absence of a general framework is not only contrasting to major trends in programmable networks [3] but also to prior developments by the TINA consortium of leading telecommunication vendors, Telco's, and their research institutes. They modeled the network and administrative facilities of a traditional telecom operator [15] in an object oriented, CORBA based [4], framework.

The details of the interfaces between applications and NEs are also application specific. Depending on their insights, developers choose different implementation technologies for a given application. As stated in section 2.4, the Ruby RMI DRb was currently considered to be more suitable to implement a NE service interface than webservices. The footprint of webservices in the NE was expected to be too large.

In this paper we investigate the consequences of the network virtualization itself. We do not want to standardize interfaces or invent brokers, directory services, etc before experience, insight and scientific studies yield reasons to do so.

4.1. The implementation UPVN

We have created several applications of network services following the UPVN concept; some of them are described in the next sections. As a result, the service architecture of Figure 5 appeared quite convenient for our applications. We have created a Virtualized Network Service (VNS) that exposes a SOAP based webservices interface. Here, its large footprint is not considered to be a problem, since the VNS runs on a dedicated computer and not on a NE with limited capabilities as in the case of sensor networks.

The combined services of the Network Utility Service (NUS) and the Network Component Collection (NCC) can be used via the VNS. The NCC provides access to specific ACs. Figure 5 illustrates that NCs have (Ruby specific) connections to the NE and more specifically to the NCSs service interface (see section 2.4). Applications and NEs can load the NEs with specific ACs during run time using the ACM (section 2.4).

The NUS was developed for applications that use path and topology functions to manipulate certain token-tagged streams. The Token Transaction Service (TTS) allows for transactions on reservation of NE packet forwarding capacity between specific incoming (ingress) and outgoing (egress) links of specific tagged IP packets. The TTS controls a specific AC for this. The Uniform Cost Search Service (UCSS) finds the least cost path between in the Network of NEs by using the AdjNe AC described in section 2.4.

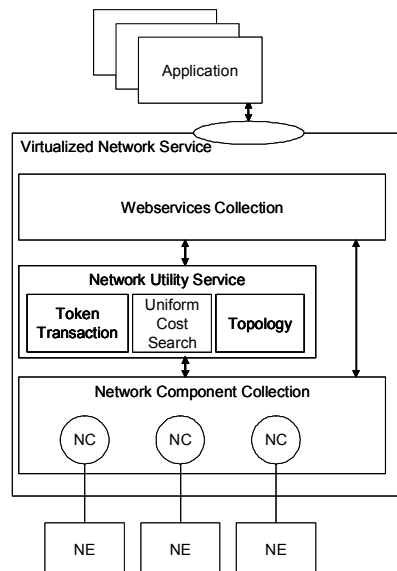


Figure 5. Architecture of the virtualized network service. This architecture is specific for topology aware applications as the ones we build in Mathematica.

AdjNe is also used by the Topology Service (TS) to find NEs and Internet access points. The TS is also able to store discovered NCs in the NCC. Furthermore, it stores topology information in objects. In this object model, each NE is characterized by its ports and links to other NEs or Internet Access Points. The object model accommodates virtual ports and links and therefore VNEs, see section 3.

4.2. The discovery of NEs and Internet access points

The network service is provided through NCs in the application. But how does the application know which NCs are available? Clearly there are many answers to this question. If the NE service interfaces are constructed with webservice technologies, the UDDI mechanism presents itself. Alternatively, in our experimental implementations NEs are registered manually at the NCC. However, one can do without a central registration of NE service interfaces. This is because that by using neighbor discovery services on NEs, a single NC suffices to discover the services interfaces of all other NEs in a connected graph, e.g. by using AdjNe (see section 2.4).

Consider the topology of Figure 2. Suppose the application contains only a reference to NE 5, then the BFS algorithm [16] would find almost all of the NEs. NEs 10 and 11 are not found. NE 10 is connected to 5 only through the Internet and cannot be discovered by AdjNe from 9. NE 11 has neither connection to the

Internet, nor to the network of NEs. For applications to interact with NEs like 10 and 11, some other means have to be used to find them.

5. UPVN Applications

In UPVN, applications might be distributed, but they will act as a single entity. Once the virtualized network is created, it is available to application developers in the form of NCs, probably implemented as objects. Then, decades of progress in computer sciences can be applied immediately to create new network aware and network centric applications.

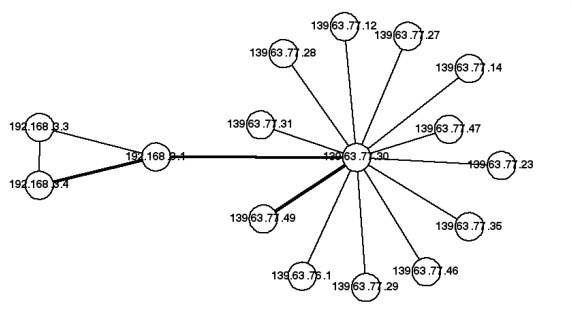


Figure 6. Network topology and visualization of the shortest path (thick line) function in Mathematica's Combinatorica package. The vertical bars at right are a typical Mathematica layout feature.

To illustrate this, we have experimented with implementations of VNS to facilitate an interface with Mathematica [17] on basis of webservice technologies. The Virtualized Network Service described in section 4.1 was the result. Figure 6 shows a network topology drawn by Mathematica. Figure 7 and Figure 8 illustrates the apparent ease of the use of VNS by Mathematica.

```
Needs["WebServices`"]
<<DiscreteMath`Combinatorica`
<<DiscreteMath`GraphPlot`
Print["The following methods are available from the
NetworkComponent:", InstallService["http://
localhost:3000/network_service/service.wsdl"];
The following methods are available from the
NetworkComponent:
{GetAllLinks, GetAllElements, NetworkTokenTransaction}
```

Figure 7. Initializing the network component webservice in Mathematica

```
n = GetAllElements[];
e = GetAllLinks[];
nids = Apply[Union, e];
Print["Network elements: ", n];
Print["Number of ports found: ", Length[nids]];

Network elements: {bigvirdot, virdot}

Number of ports found: 16
```

Figure 8. NE discovery in Mathematica

This combination of Mathematica and UPVN shows a glimpse of entirely new networked applications. It illustrates also the enormous wealth of mathematical, computational and visualization tools that can be unlocked for specific applications using generic technologies made available via Grid and webservice interfaces.

5.1. Paths and applications

A whole body of graph theory is at the disposal of the application developer to develop algorithms that satisfy particular purposes. Using flow calculations on basis of actual and near real time data, one can judge how much network capacity is unused. For instance, with a uniform cost search algorithm, it is straightforward to develop an application that finds a least cost route, with a minimum bandwidth guarantee. More sophisticated algorithms allow faster results and faster run time, albeit requiring more understanding from the application developer.

```
nodePath =
ConvertIndicesToNodes[ShortestPath[g, Node2Index[nids, "192
.168.3.4"], Node2Index[nids, "139.63.77.49"]], nids];
Print["Path: ", nodePath];
If[NetworkTokenTransaction[nodePath, "green"]==True,
Print["Committed"], Print["Transaction failed"]];

Path:
{192.168.3.4, 192.168.3.1, 139.63.77.30, 139.63.77.49}

Committed
```

Figure 9. Example of path provisioning in UPVN.

Figure 9 shows the determination of a shortest path with Mathematica's `ShortestPath` function from the `Combinatorica` package. Since our NEs support transaction services, one can write applications where the reservation of a path elements, a video movie and a pizza is committed if all of them are available in a given period!

Figure 9 shows also a Mathematica code snippet that invokes a transaction to claim all nodes along a shortest path. The transaction is successful if all key provisioned nodes decode the token "Green". One part of this token has been left on the node as a proof of a successful negotiation of AAA matters [11] for specific AC packet services with the owner of the NE. Developments like these bring the powerful facilities of transaction monitors like CICS [18] in the realm of applications that create application specific paths on nation wide network infrastructures for special purposes.

In the case of OSI and IP protocols, algorithms on routers take care of proper forwarding of packets. In case of failures, the routing protocols automatically

find alternative routes, if they exist. IP routing is reactive, it responds to a change. In IP networks a routing reconfiguration as a response to a failing router can cost many seconds. For VoIP applications this would result in noticeable effects. In UPVN, applications are aware of the network state and can anticipate problematic situations. ACs in NEs could carefully test and monitor the health of a single or a group of NEs (e.g. along a path) to the point that a malfunction is detected before or immediately after it occurs. Applications could then instruct NEs to change their routing tables and their routing policies to forward packets along alternative paths, if they exist, that satisfy application needs. Furthermore, by a modification¹ of the uniform cost algorithm, one can easily find (if they exist) the least cost N alternate paths between a given source and destination, yielding another strategy to find robust paths. In sensor networks the alternate path method not only adds to the reliability of sensor telecommunications, but also distributes the power consumption more equally over the nodes. This is, in many cases, a good strategy to increase the deployment time of the sensor infrastructure that uses batteries.

5.2. Topology changing applications

In some applications, topology is a concern and application programs have to deal with this. Once some or all NEs are known, one is indulged to make a map of their topology. However, we do not expect programmers to write programs for specific topologies. The application designer mostly does not, and cannot have detailed knowledge of the network. Application programmers rarely construct networks. In the case of wireless sensor networks, optimal application topologies may vary constantly. To deal with networking issues, application programmers develop rather algorithms to deal with application specific network issues.

In the previous section, we have shown how NEs in a network can be found, how paths can be established in an optimal way and elaborated on strategies by which applications can maintain a certain service level from one or more paths between source and destination. There is another situation, one that we explicitly want to deal with, in which application developers are not aware of the actual network topology. This situation occurs frequently in wireless sensor networks where nodes continuously join and leave the network during application run time. As an example, consider a situation in which occurrences of articulation vertices, a single node whose failure can isolate part of the net-

¹ Find the least cost path by the uniform cost method. Remove all NEs present in the least cost path between source and destination from the search input. Rerun the algorithm to find the next least cost path.

work, are unwanted. The Mathematica program listed in Figure 10 finds these vertices with the function `ArticulationVertices` from the `Combinatorica` package.

```
ConvertIndicesToNodes[ArticulationVertices[g], nids]
{139.63.77.30, 192.168.3.1}
```

Figure 10. Mathematica's detection of articulation vertices.

If sensor NEs would have the ability to create new connections, e.g. by increasing the emitting power of its transmitter or by adjusting the directional sensitivity of its antenna, the NEs could be instructed to change these parameters until Mathematica calculates that articulation vertices have disappeared.

6. Application configurations

Until now we did not elaborate on where the applications run. A very plausible configuration is one in which the (distributed) application connects via the NCs to the NEs. At this point we say that the application is external to the network.

The NCs may only be available on the NE. Therefore the application itself must then be contained in the NEs. In the special case of a single application, it has to travel from NE to NE to interact with each NC; the application becomes an agent [5]. Traveling here is the process where the NE sends to another NE the application code and the persisted state of the application. VMware has the ability to freeze an entire operating system with running applications and to send the frozen system to another computer. There the execution of operating system and everything it runs is continued. In collaboration with Nortel, we demonstrated this ability at SuperComputing 2005 [19] by moving a Linux operating system, running a picture database search application, between computers located in Amsterdam, Chicago and in San Diego.

There are various ways to implement communications between the NC in an application and its NE. Clearly, in some situations one could design a separate network that prevents unwanted interactions of ACs with NC-NE communications. In other cases, such a separate network is not possible and one could use certain ACs to increase the reliability of NC-NE communications. One could for instance design an AC that makes the transmission of information destined to ACs more reliable by implementing a logging and store-and-forward mechanisms. Here one could copy some of the technologies message queuing deploys. With the use of tags containing secured tokens, the NE can discern the appropriate packets in the incoming streams

easily. To this effect, NEs may fetch appropriate ACs such as performed in discrete Active Networks [20].

7. Conclusion

UPVN shows that well known concepts for programmable networks augmented with insights from Grid leads to new and practical applications. UPVN applications can deal, by programming the NEs, optimally with changing Internet conditions.

The UPVN concept is foremost useful in situations where structural tuning of applications and network nodes is not feasible any more. Trivially this occurs when many NEs in many network domains are involved. Such situations occur also when networks and applications run close to their maximum capacities and financial budgets – continuous tuning is necessary. Furthermore, we showed UPVN to be equally useful in future mobile sensor network applications, where the communication facilities need to be tuned regularly to the changing state of the sensor nodes.

We thank dr. L. Torenvliet of the University of Amsterdam for discussing graph theory applications.

8. References

- [1] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, New Jersey, 2003
- [2] www.cisco.com/application/pdf/en/us/guest/products/ps6776/c1650/cdcont_0900aecd80455385.pdf
- [3] www.iwan2005.net
- [4] Andrew S. Tanenbaum, Maarten van Steen, *Distributed Systems*, Prentice Hall, New Jersey, 2002
- [5] www.ietf.org/html.charters/forces-charter.html
- [6] Christos Chrysoulas, Evangelos Haleplidis, Robert Haas, Spyros Denazis, and Odysseas Koufopavlou, "A Web-Services Based Architecture for Dynamic-Service Deployment", *International Working Conference on Active and Programmable Networks*, Sophia Antipolis, France, 2005
- [7] Evangelos Haleplidis, Robert Haas, Spyros Denazis, and Odysseas Koufopavlou, "A Web Service- and ForCES-based Programmable Router Architecture", *International Working Conference on Active and Programmable Networks*, Sophia Antipolis, France, 2005.
- [8] Gísli Hjálmtýsson, Keynote talk: "Architecture Challenges in Future Network Nodes", *International Working Conference on Active and Programmable Networks*, Sophia Antipolis, France, 2005
- [9] Eric Bonabeau, Marco Dorigo, Guy Theraulaz, *Swarm Intelligence*, New York: Oxford University Press, 2003.
- [10] Charles E. Perkins, *Ad Hoc Networking*, Addison-Wesley, 2001.
- [11] Leon Gommans, Cees de Laat, Robert Meijer, "Token Based path authorization at Interconnection Points between Hybrid Networks and a Lambda GRID", *IEEE GRIDNETS2005 proceedings*, ISBN 0-7803-9277-9.
- [12] www.read.cs.ucla.edu/click/
- [13] S.M.C.M. van Oudenaarde, Z.W. Hendrikse, F. Dijkstra, L.H.M. Gommans, C.T.A.M. de Laat, R.J. Meijer, "An Open Grid Services Architecture Based Prototype for Managing End-to-End Fiber Optic Connections in a Multi-Domain Network" in *High-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project, Special Issue Future Generation Computer Systems*, Volume 21, Issue 4 (2005).
- [14] Ruby: www.ruby-lang.org
- [15] TINA: ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/35/15499/00714634.pdf?arnumber=714634
- [16] T.H. Cormen, C.E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, Cambridge: The MIT Press, New York: McGraw-Hill Book Company, 2000, pp. 470
- [17] Mathematica: www.wolfram.com
- [18] CICS : www-306.ibm.com/software/htp/cics/
- [19] F. Travostino, P. Daspit, L. Gommans, C. Jog, C.T.A.M. de Laat, J. Mambretti, I. Monga, B. van Oudenaarde, S. Raghunath and P.Y. Wang, "Seamless Live Migration of Virtual Machines over the MAN/WAN", *iGRID2005 special issue, Future Generation Computer Systems*, volume 22 issue 8, pp. 901-907 (2006).
- [20] David L. Tennenhouse, David J. Wetherall, "Towards an Active Network Architecture, keynote", *Multimedia Computing and Networking conference*, San Jose, CA, January 1996.