Interactive Networks

A User Interface for Application Owned Lightpath Networks

Editor's Note: The prototype discussed below was developed by Rudolf Strijkers under supervision of Prof. Dr. Robert Meijer as part of his phd research on next-generation Internet architecture. Members of the team that developed the demo were Mihai Cristea (post-doc), Laurence Muller (scientific programmer) and Robert Belleman (head of UvA visualization lab). My Interview with Rudolf was conducted on November 19.

Figure 1. Interactive Networks setup at Super Computing 2008, Austin, TX.

COOK Report: Cees de Laat and I've been down in the engine room talking about what makes these user controlled and application controlled switched light path networks possible. Now we are going to look at a prototype of a user interface with a multi-touch-sensitive screen that allows a user to tap on the tools he wishes to select and with his finger to draw the paths he wishes to activate nodes on a programmable network. Multiple researchers can use the multi touch interface at the same time.

According to what you were telling me the goal of what we are talking about is to have this kind of software on an ordinary researcher's screen three or four or five years from now. At that point the user should be able to use it to control most any application in collaborative environment. Would you take me on a guided tour of what it is and what it does and how it works?

Strijkers: This is the first prototype of what we are calling "Interactive Networks". In interactive networks humans become an integral part of the control system to manage the next-generation of programmable networks and Grids. The main design principle is this; by virtualizing the configurable and programmable properties of network elements as software objects, any aspect of a network infrastructure can be manipu-

lated from computer programs. What we show here is an implementation of an interactive control system concept for user programmable networks, which applies the architectural concepts we have developed in our research.

The network you see here is the current set up of our test bed located at the University of Amsterdam. These icons represent the network's elements and network structure visualized at IP level. I can tell you a little bit about the actual infrastructure. There are currently twenty nodes, interconnected by four subnets to create an operationally interesting topology. Three of these subnets are in separate virtual machine environments of VMware, called ESX servers, that also contain four virtual machines each. ESX is essentially a container for virtual machines. The VMware management environment enables us to create, clone, and remove virtual machines. It also enables creation and manipulation of complete virtual network environments.

The virtual subnets and machines are connected to a physical subnet, which also contains two physical nodes. Then we have the four Mac Mini's here in the booth, which are directly connected to the physical switch in Amsterdam with a gigabit connection. This way, the Mac mini's are part of the subnet.

When the network boots and the nodes come up they will connect to controller. The controller is programmed in such a way that it will send a out a discovery request when a new node connects. Each node will try to discover its neighbors using ARP to scan the whole subnet for hosts. Since it will discover everything in the subnet, we also find all the nodes in the datacenter. We currently only display the programmable nodes, but the whole discovered network can be displayed too.

At SC08 we forgot to turn off WiFi to Scinet on the Mac mini's once and discovered over 700 nodes within seconds. The neighbors you discover at Ethernet level will look like fully connected at the IP level. For example, if you interconnect the three computers with a switch, it will always look as though each computer can reach each other directly. That's why it looks like three fully connected networks here and one large fully interconnected network over there; the video screens are just one hop away.

Once we have discovered the networks you can also see the result over there in the Mathematica interface on the fifth screen. The Mathematica interface has access to the same information and supports the same network ma-

Figure 2. Network visualization after discovery. The dashed lines, when monitoring is enabled will indicate bandwidth (line width), delay (percentage of dashing) and jitter (randomize in dashing).

nipulations as the touch table. We will come back to this aspect later.

COOK Report: What do these icons represent?

Strijkers: The icons represent the type of modes or functionality that each node offers. We currently have three modes. 1. A producer: this node contains streaming video content and is visualized as a green-circled arrow. It can also route traffic. 2. A consumer: such a node is connected to a streaming video client and can display the streamed content. A play button in a screen shows it. 3. A router: The sole purpose of these nodes is to route traffic and blue-circled arrows demonstrate it.

At this point we can look at what kind of videos may be streaming in the network. You can push with your finger on a producer node and a window opens on the touch table that gives you a preview of the video. By the way, the movies we currently have are: Big Buck Bunny, Elephants Dream, and two Cinegrid demo movies. The first two are actually made in the Netherlands as part of the Orange Open Movie Project and the son of Cees de Laat made one of the CineGrid movies. All the movies stream continuously stream in high definition, but to unknown destination IPs. This way a node never receives a stream, except when we run our special expressions to capture the traffic.
Underneath the node you can see a small graph, which will displays CPU load measurements. A button on the top right of the touch table will enable or disable the CPU load measurements in the network. The real-time load information for a selected node will be displayed in this graph. When the load becomes larger than 1, the nodes will light up red, alerting the operator that the node is under stress.

Now we can decide to make a path. Then we go into the path creation mode, and then you can decide to trace a path, from a producer node to a router to a screen, for example. And you see the video stream appear on the screen.

COOK Report: The blue line is traced with the finger? [See Figure 4 on page 44 below.]

Strijkers: Yes. If we make a path, what will happen, we send a request to the controller asking to create the path that we just dragged with our finger. The controller will send the request to a compiler to generate the commands and forwarding expressions for provisioning the nodes. The results will be passed on to a transaction monitoring, which executes a distributed transaction to load the commands and expressions on the nodes. If loading of one of the expressions or commands fails, the whole transaction will be

Figure 3. By touching a producer node you will see a preview of the video stream. The small graph underneath shows real-time CPU load measurements.

Figure 4. Simply drag a line with a finger to create a path.

rolled back. So, whatever happens in inserting or removing requests, the network will always be in a consistent state. When drawing a path, the touch table will also automatically select an unused color and this color will be used for lifetime of a path to identify the stream. When tracing the traffic on IP level you will actually see the color codes in the IP packets.

Once we have a path we can also select it and extend it to a multicast path. This can be done by dragging a new route starting from any node of the path chosen for the extension. It's as easy as that.

COOK Report: You are taking this content and sending it to a second screen?

Strijkers: Exactly. I only have to touch the path on a node and drag it to a screen over another route. And then you can see the movies streaming on two computers.

COOK Report: With this table, and if somebody will show me, I'm sure I will learn the basics of it pretty quick, right?

Strijkers: Yes, you will. The interface is very simple considering the capabilities. Actually, what we show here is a showcase for how we envision the management of the next-generation networks. So, the capabilities of our experimental programmable network are one step further than what you can do with modern networks now. For example, I can show you how we can draw a path with a loop. Have you ever seen a path with a loop in a network?

COOK Report: So, in other words, that path that you just set will send data back and forth two or three times?

Strijkers: Yes, the packets will ping-pong back and forth before being routed to the screen. Normally, loops in a

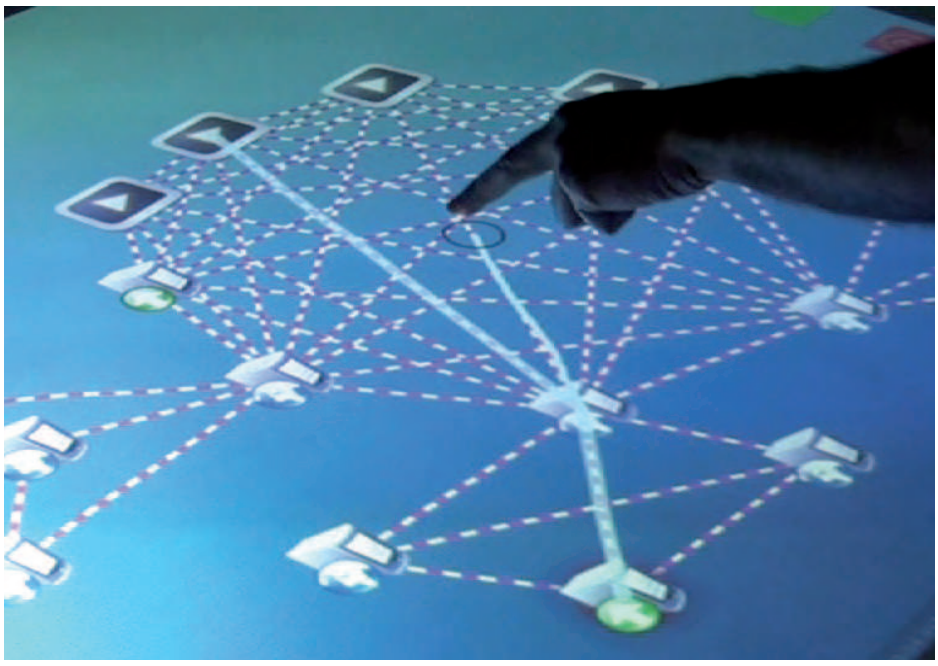

Figure 5. Selecting and extending an existing path from a node creates multicast paths.

Figure 6. Creating a loop. Unicast paths can be routed any way a user likes, whether it contains one or more loops or crossings. The compiler will detect and generate the correct expressions.

network are bad, because routers have no way of detecting if a certain packet already came by or not.

To achieve loops in a programmable network, you could use special programs with counters to detect looping packets. But, we enhanced IP a little; we put a token inside the packet, in the IP option field to be more precise. This token is not necessary, but it allows us to white list or identify packets or streams uniquely. The token enables us to bind network behaviour to traffic that is not in any sense connected to the protocols used. An example of such a binding is 'I want a good quality video connection to my TV, but only after my pizza arrives', but also to bind network behaviour of communities or distributed applications in grid networks. Our former colleague Leon Gommans did a lot of work on this subject and we have extended his ideas to programmable networks.

In the demo, the compiler detects the loop and creates expressions that change the colour of a token to a different shade. This means that at every hop the token is rewritten and will flow in another streamline graph. For example the colour still remains blue to the application, but in reality the network uses the shades to maintain state.

So, I have shown how to make a unicast path, how we made a multicast path, and how we made a path with a loop. Now I can show you a little bit what happens inside the node.

COOK Report: *OK*

Figure 7. Changing path creation mode. The button with the question mark shows the discovered topology including non-programmable nodes.

Strijkers: For this we need to switch to a different mode. This mode disables dragging of paths and allows us to interact with the nodes and edges of the network. When I double tap on a node, you will see what happened in the node when we made the loop.

COOK Report: The large black circle indicates you zoomed into the node?

Strijkers: Yes. When we make a path, like a loop for example, the request is sent to a controller. This controller runs the request through the compiler, which checks if the nodes are available, how they are connected, if the source is a video stream and if the destination is connected to a screen. The compiler will generate a flow graph for every node. This flow graph will describe how the traffic flows from the input ports to the output ports.

COOK Report: And the output port is **skb** in the red circles?

Strijkers: Yes

COOK Report: And the tbs, what does that stand for?

Strijkers: Let's start at the first filter first. Netfilter is a library in the linux kernel, which captures all the data from the networking stack at specific points. When not used, traffic would normally go through the normal networking stack of Linux. But, what we do is we capture the traffic at Netfilter input and force it to go through the flow graph. And, skb_transmit is actually the output function of the linux kernel. So, if we send a packet there, it will be routed and sent to the correct host. We have made a special modification, were we have full control over the traffic flow. This first filter is tbs. It's called the token based switch. What it does is, it looks at the packet and says, it's a blue packet, and I'm a blue graph, so I accept the packet. If I would be a red graph I wouldn't accept the blue packet. In other words, it accepts or drops packets based on their token. This allows us to create application-specific flow graphs for tokenized streams.

When the packet is accepted, it goes on to the tb and the tb filter tears off the token. Why would we tear of the token? Because if we send the message to the Mac mini, it would have no clue that we did all kind of weird stuff with the packet. It just looks like any other packet. And here is the magic; we have a filter that rewrites the IP destination of the packet to route it to go there. The library that allows us to insert/remove these filter expressions at run-time in the kernel is called Streamline. It was developed at the Vrije Universiteit of Amsterdam and our colleague Mihai Cristea and I worked closely with its developer Willem de Bruijn to make it suitable for our purpose.

Figure 8. Streamline flow graph currently loaded in a node. Note that due to the picture contrast the connecting lines are barely visible.

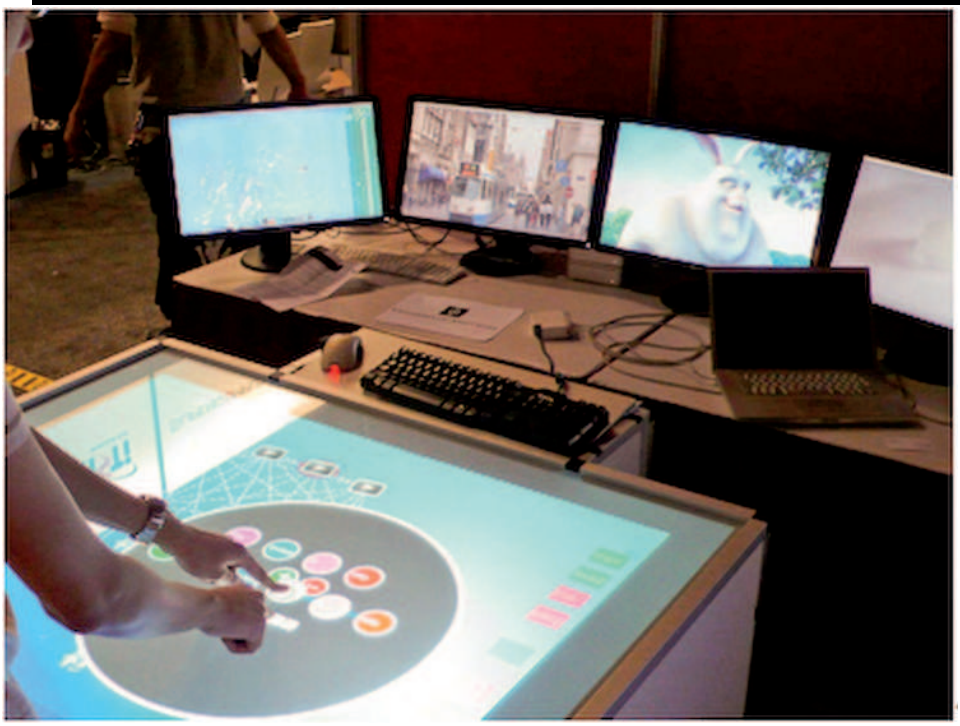

Figure 9. Modifying the sampling rate of a flow. The extreme left shows the manipulated stream, and the screen immediately to its right shows an unmodified stream that is also part of the same multicast tree.

COOK Report: When you get a green circle with a plus, what does that indicate?

Strijkers: The compiler automatically generates these expressions, the distributed transaction processor executes a two-phase commit on all the nodes and inputs them in Streamline. After the transaction is complete, we can zoom into the node and modify the expression that resulted from the compilation process and that is currently running in Streamline. Just by touching the plus button.

COOK Report: That gets you to a different interface, or?

Strijkers: Not exactly, it adds a filter to the run-time expression at a certain place. Keep in mind that the actual code is running in the kernel.

What actually happens is that this request goes to the specific node, it picks out the manipulated expression, plugs the sampler in and puts it back into Streamline. And you can see it, because your whole video stream goes nuts. On the left screen we see the video of a multicast branch with a sampler and on the right the unmodified stream of the other branch.

The image is distorted because we throw away some packets. Right now it throws away 50 percent of the packets. (Modifies the sampler value) So, you just saw me modifying the flow dropping rate in real-time. Now we only implemented user interface support for a sampler, but you can insert any type of filter yourself or even write your own.

This is a powerful tool to exert very fine-grained control over traffic. For example you could add or manipulate rate limiter filters, which would allow very precise traffic shaping. The operator at the touch table could manually control the traffic shaping, but it is also possible programmatically. I can show this later on.

Strijkers: And we can say, hey, we change the sampler to 80 percent. You will see that the screen gets gradually better. It's because of the encoding that it will do weird stuff. But, we can also remove it with the minus. It will go streamline again, it will remove the sampler and you will see the stream turning back to normal.

COOK Report: Impressive!

Running Mathematica

Strijkers: So, that's the touch table part. The other part is we can enable throughput measurements, so then the controller will ask to the network to continuously return throughput

Figure 10. A closer view on the sampler modification interface. The sampling percentage is changed with the slider.

measurements. And you can see what happens over there.

COOK Report: You're over at Mathematica there

Strijkers: Yeah, Mathematica is a scientific computing environment. It allows you to do interactive calculations. Mathematica contains a large library for statistical analysis, graph theory and so on. And by the way, today they released version 7, which includes out-of-the-box support for parallel computing. Mathematica also is a powerful symbolic language that allows you to write programs and dynamic visualizations. I implemented a Mathematica interface to the programmable network and now Mathematica is part of the programmable network and is able to receive measurements or manipulate paths just like the touch table.

COOK Report: You can see the orange yellow cone changing shape, and it looks almost like the visualisation of a pumping heart. That's my metaphor.

Strijkers: It's a 3D contour plot of the real-time throughput in the network, so the analogy with the pumping heart is quite accurate. The contour plot shows that we can now directly apply the large collection of Mathematica libraries to visualize

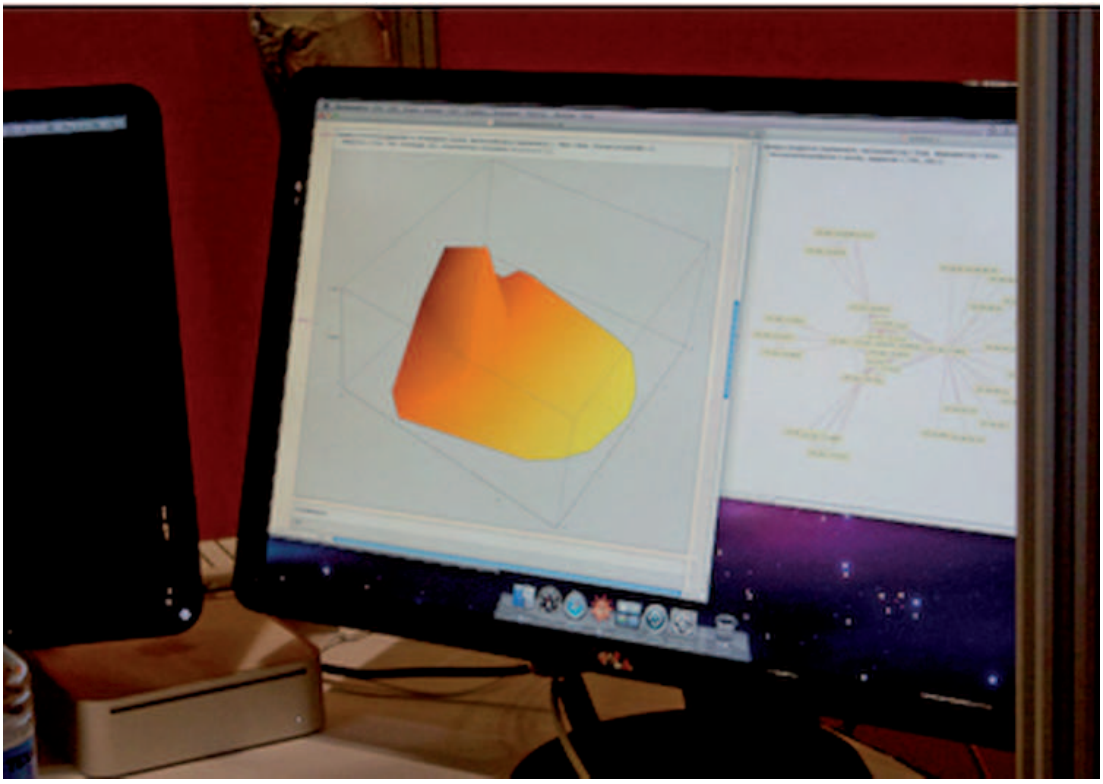

Figure 11. Interactive Networks in Mathematica. The left window shows the 3D contour plot of the real-time throughput in the network. The right window shows the current topology of our programmable test bed.

and program networks interactively while using real-time measurements. On the touch table we do graph layouts by hand for example; Mathematica can calculate the layout automatically, which you can see in the other window. To create the dynamic plot that is shown on the screen, a graph layout algorithm of Mathematica determines the form of the surface. Then the throughput measurements make up the values of the z-axis.

With the information given by the network, and currently we support continuous measurement of delay, jitter, bandwidth and throughput, one or more operators at once can write programs that automate decision-making. This opens the way for automated net work adaptation in a user-friendly environment.

For example, now we can say, certain paths should avoid busy parts of the network. By using only standard functions in Mathematica, it is already possible to write a simple program that uses the real-time throughput information to continuously reroute one or more paths that avoid busy parts of the network.

COOK Report: In three years time, this software and capability should be on every researcher's workstation?

Strijkers: We certainly hope so. The demo we showed today illustrates a novel way to manage programmable networks. But, it is not limited to this case only. **Amongst other things, we would like to incorporate light-path management and include other resources, such as storage or virtual machine management. We hope that on a longer term we can do trials on a larger scale, for example, by integrating our solution with Internet2, ARGIA and other research networks and Grids. Eventually, we hope to see our work applied in collaborative control room environments for monitoring and control of complex and large-scale systems.**