NetServ: Dynamically Deploying In-network Services

Suman Srinivasan*† Jae Woo Lee*† Eric Liu*† Michael Kester*†
*Suman, Jae, Eric and Michael contributed equally to this paper.
Henning Schulzrinne† Volker Hilt‡ Srini Seetharaman§ Ashiq Khan¶

†Department of Computer Science, Columbia University, New York, NY, USA
{sumans,jae}@cs.columbia.edu, {ewl2113,msk2117}@columbia.edu, hgs@cs.columbia.edu
‡Bell Labs/Alcatel-Lucent, Holmdel, NJ, USA
volkerh@alcatel-lucent.com
§Deutsche Telekom R&D Lab, Los Altos, CA, USA
srini.seetharaman@telekom.com
¶DoCoMo Communications Laboratories Europe, Munich, Germany
khan@docomolab-euro.com

ABSTRACT

We present NetServ, an extensible architecture for core network services for the next generation Internet. The functions and resources available on a network node are broken up into small and reusable building blocks. A new core network service is implemented by combining the building blocks, and hosted in a sandbox-like execution environment that provides security, portability, resource control, and the ability to deploy modules dynamically.

We describe our first prototype, a novel combination of the Click router and the Java-based OSGi module system. Our measurement results indicate that the processing overhead incurred by the Java layer is a reasonable trade-off for the level of modularity we achieve in our system.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Architecture and Design*

General Terms

Design, Measurement, Performance

Keywords

FIND, Java, OSGi, future internet, modular, netserv, programmable, routers, service oriented architectures

ReArch'09, December 1, 2009, Rome, Italy.

1. INTRODUCTION

Despite the tremendous success of the Internet in the past decade, a number of shortcomings of current Internet architecture have become apparent. The *ossification* of the Internet, often suggested as the main problem of the current architecture, refers to the fact that it is nearly impossible to add new functionality and services to the network core. This is clearly shown by the dismal rate of adoption of new Internet protocols such as multicast routing and QoS, even when the need for them is widely recognized. Many have naturally turned to implementing network services on the application layer using overlay networks formed by end hosts, since providing services through overlay networks eliminates the need to update the network core. However, such solutions tend to be ad hoc, often duplicating the effort of other overlay networks, and inefficient because certain basic functions such as distance estimation can be achieved much more effectively at the network core.

We present NetServ [10], our on-going research effort to design an extensible architecture for core network services for the next generation Internet. The key idea of NetServ is *service modularization*. The functions and resources available on a network node are broken up into small and reusable *building blocks*. A new core network service can then be implemented by combining the functionality of building blocks available on multiple network nodes. We use the term *service modules* to refer to the building blocks or the composite components that use multiple building blocks.

Another piece of the NetServ architecture is the *virtual services framework*, which refers to the architecture of the network nodes that host service modules. The virtual services framework provides a sandbox-like execution environment for the service modules, offering security, portability across hardware platforms, and the ability to control resource allocation among modules. In addition, the framework supports adding and removing service modules at runtime, by network administrators or even by content providers and end users, enabling on-demand and per-flow services in the network core.

In this paper, we describe our first prototype implementation of the NetServ architecture. We used the Click modular

router [19, 3] as a base router platform, and augmented it with a Java-based dynamic module system called OSGi [13], which provided the ability to load and unload service modules at runtime. Currently, the prototype implements a component inside the Click router that intercepts incoming packets and sends them to Java service modules that can be installed and uninstalled at runtime. The Java technology provides portability across hardware platforms, and a comprehensive security framework on which to build our security and resource control mechanisms in the future.

The Java technology has not been used much in routers, most likely due to concerns on its performance. We compared our prototype with a Click router running as a user process and also with a Linux kernel acting as a router using Maximum Loss Free Forwarding Rate (MLFFR) as the metric. Our results indicate that the processing overhead incurred by the Java layer is quite acceptable, as it is much smaller than other unavoidable penalties associated with systems where modules can be installed dynamically.

The rest of this paper is organized as follows. Section 2 gives a brief overview of the two technologies that we have used to implement our prototype: the Click modular router and the OSGi framework. Section 3 describes our prototype implementation in detail. Section 4 presents our measurement methods and results. Section 5 summarizes the related work. Finally, we conclude and discuss future work in Section 6.

2. TECHNOLOGY OVERVIEW

2.1 Click Modular Router

Click [19, 3] is a modular software architecture for Linux and other UNIX-like platforms that allows for the creation of easily reconfigurable routers and switches. Click functionality is manipulated using a text file that specifies how modules, called *elements*, are arranged in a directed graph. The graph structure allows for numerous possibilities. One such possibility is shown in Figure 1. This example of a very simple Click configuration receives packets from eth0, counts them, and discards them. Click includes hundreds of predefined elements so it is easy to reconfigure a graph to implement many types of network devices. In addition, custom elements can be written to further extend functionality.

The Click router can run in two modes: user-mode or kernel-mode. User-mode runs as a user-level process. This means it does not replace the routing performed by the underlying Linux kernel. In contrast, kernel-mode runs as a module inside the Linux kernel and can replace the routing functionality of Linux.

The performance of the Click router is much higher in kernel-mode, and it can be further enhanced by replacing the standard Linux Ethernet drivers with polling drivers. Polling drivers turn off Linux's interrupt structure and device handling, and allow the network card to poll for packets.

Kernel-mode uses the `proc` file system to access data from a running element or to change the element's settings. If more extensive changes are required, Click offers the ability to replace the running configuration with an entirely new one, called *hot-swapping*. Compared to NetServ, Click's hot-swap feature is limited in three ways. First, Click elements are written in C++, thus the elements in binary form can be installed only on a particular hardware platform. Second, Click elements run inside the kernel so there is little to no

```
FromDevice(eth0) -> Counter -> Discard;
```

Figure 1: A minimal Click configuration.

security or access control. Third, the ability to hot-swap a particular element into a running Click router is dependent on the element having been compiled into the Click kernel module. The router must be restarted to insert a newly developed element.

2.2 OSGi Framework

OSGi™ [13] is a component framework for Java. In the OSGi framework, an application is organized as a set of modules, called *bundles*, which are Java Archive (JAR) files [6] that conform to the structure specified by the OSGi framework. The bundles can be loaded and unloaded at runtime. This enables installing a new feature into a running application or upgrading a part of it with newly written code, without having to shutdown and restart the application. There are a number of implementations of the OSGi framework available today, including open source software such as Apache Felix [1] and Eclipse Equinox [4].

In a normal Java application, a class can usually access any other public class in the same application, i.e., a class can create an instance of another public class and invoke a method on it. In OSGi, the scope of such an unrestricted access is limited to the enclosing bundle. In other words, the classes that belong to a bundle are *not* visible to the other classes that belong to other bundles. The only method of inter-bundle communication is for a bundle to explicitly *export a service* by listing a package containing the interfaces in the manifest file of its JAR file, and for another bundle to explicitly *import* the service, also by using its manifest file. The OSGi framework achieves this isolation of bundles by using a custom class loader.

3. NETSERV IMPLEMENTATION

We implemented a prototype of NetServ using the Click modular router as the base platform. The Click router provides a high performance router platform that can be easily extended because of its modular approach. Extending the Click router is a matter of writing a new C++ class–an *element* in Click terminology–that extends a simple base class with a few member functions. This enabled us to develop our prototype concentrating on the NetServ functionality without having to worry about the basic router functionality. The current version of NetServ is based on the user-mode Click. Implementing NetServ on the kernel-mode Click is planned as a future work.

On top of the Click router platform, we used the OSGi framework. OSGi provides an ideal foundation on which we can realize our vision of a secure and portable services framework that supports dynamic distribution of services. Since OSGi is based on Java, it naturally inherits the portability across hardware platforms and the comprehensive Java security architecture [7]. OSGi's ability to load and unload bundles at runtime satisfies the fundamental requirement of dynamic distribution of services. The strict separation of OSGi bundles provides a solid starting point to address the security concerns associated with dynamic distribution of services.

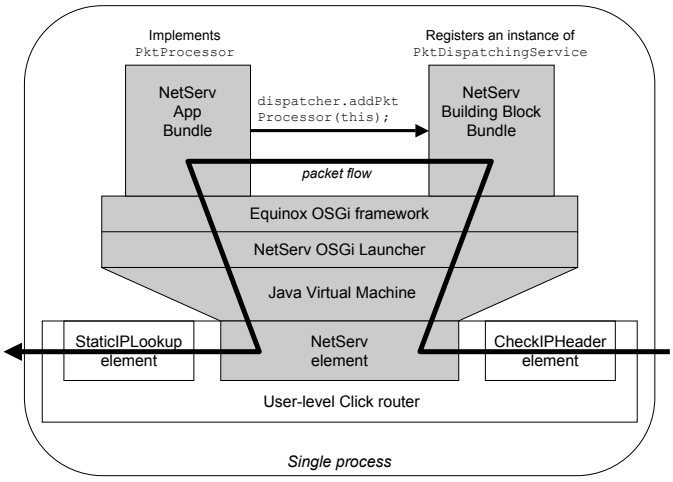

Figure 2: NetServ prototype architecture.

Figure 2 depicts the overall architecture of the prototype implementation. The shaded boxes represent different components of NetServ, and the thick arrow represents the flow of a packet being forwarded by the router, taking a detour into the NetServ components.

We wrote a Click element in C++, called NetServ, and configured a Click router to place the NetServ element on the path of the packet flow. Our test configuration was based on the basic IP router configuration that came with the Click software package. In that configuration, the NetServ element was placed between the CheckIPHeader and StaticIPLookup elements. When a Click router is started, it calls the `initialize()` member function of each element. NetServ's `initialize()` creates a Java Virtual Machine (JVM), launches the OSGi framework, and loads the configured bundles.

The NetServ element creates a JVM using the Invocation API, which is a part of the Java Native Interface (JNI) [8]. The JNI specification provides various ways for Java code and C/C++ code to call each other. The Invocation API, in particular, makes it possible for an application written in C/C++ to embed a whole JVM in the same process. After creating a JVM, the NetServ element invokes a Java function to run inside the JVM. That Java function is an entry point into the `NetServ.launch` package, represented as a box labeled NetServ OSGi Launcher in Figure 2.

The NetServ OSGi Launcher serves two purposes. First, it launches the OSGi framework, which in turn will load the NetServ Building Block Bundle and all configured application bundles. Figure 2 shows only one application bundle loaded–labeled NetServ App Bundle–but multiple application bundles can be loaded as well, in which case the packet will travel through each bundle in the order they were loaded. An application bundle implements the `PktProcessor` interface, and registers itself with the global packet dispatcher in order to receive the incoming packets. The global packet dispatcher is a singleton object which is exported as a service by the NetServ Building Block Bundle.

Second, the NetServ OSGi Launcher provides a Java class called `PktConduit`, which is visible from the Building Block Bundle and also accessible from the C++ code in NetServ element. The `PktConduit` class therefore acts as a bridge between the Java and C++ regions. Such a bridge is necessary because an OSGi bundle is loaded using a custom class loader, making it invisible to other bundles or any other code outside the OSGi framework.

The NetServ element's `initialize()` function, before it returns, also finds and saves a handle to the `injectPkt` method of the `PktConduit` class using JNI. After the initialization is completed, the NetServ element diverts every incoming packet to the Java components by calling `PktConduit.injectPkt()`, which in turn will hand over the packet to the Building Block Bundle, which in turn will invoke all registered packet processors.

We avoid copying a packet when it is passed from C++ to Java. We construct a direct byte buffer object that simply points to the memory address containing the packet using the `NewDirectByteBuffer()` JNI call. The reference to this object is then passed to the Java components.

4. EVALUATION

We want to ensure that our goal of increased modularity for network services does not come with unacceptable tradeoffs. There is a performance penalty associated with the detour that packets take into the Java layer of NetServ. We show that the performance penalty is a reasonable tradeoff. We measure NetServ's maximum loss free forwarding rate (MLFFR), which is defined to be the maximum number of packets that a router can forward without incurring any packet loss. We compare NetServ's performance to a plain Click router running in user-mode (Plain Click) and also to a Linux kernel acting as a router (Bare Linux).

4.1 Measurement Environment

The configuration we used for measurements involves three PCs, referred to as node 1, node 2, and node 3. Node 1 is connected directly to node 2, and node 2 is connected directly to node 3. When testing with only two machines, node 1 is connected to node 3 through a 10/100Mbs Ethernet switch. Node 1 is always the sending machine, node 3 is always the receiving machine, and node 2 is always the router.

The hardware used for testing was consistent but not identical for each node. Node 1, the packet source, is a Dell Optiplex 755, with an Intel Core 2 Duo E6750 at 2.66GHz, 2x 2GB DDR2 RAM at 800MHz, and an Intel 82566DM-2 Gigabit Ethernet card.

Node 2, the router, is a custom built machine with 2x Athlon MP 1600+ at 1.4GHz, 2x 512MB DDR RAM at 266MHz, and two Ethernet cards–an Intel 82557 100Mb card (eth0) and a DEC Tulip 100Mb card (eth1). We disabled one of the two CPUs in order to localize the load on the system.

Node 3, the destination, is an IBM ThinkPad T61, with an Intel Core 2 Duo T8100 at 2.10GHz, 1x 2GB DDR2 RAM at 667MHz, and an Intel 82566MM Gigabit Ethernet card.

The connections were all run at 100Mbps. All three PCs were booted into single-user mode to disable superfluous background processes. We enabled kernel-level logging for data collection.

We used kernel-mode Click to ensure that node 1 and node 3 are capable of sending and receiving as quickly as possible. For node 2, we used user-level Click since NetServ currently only runs in user-mode. Node 1 and node 3 ran Ubuntu Linux 8.04 with a patched 2.6.24.7 kernel, polling

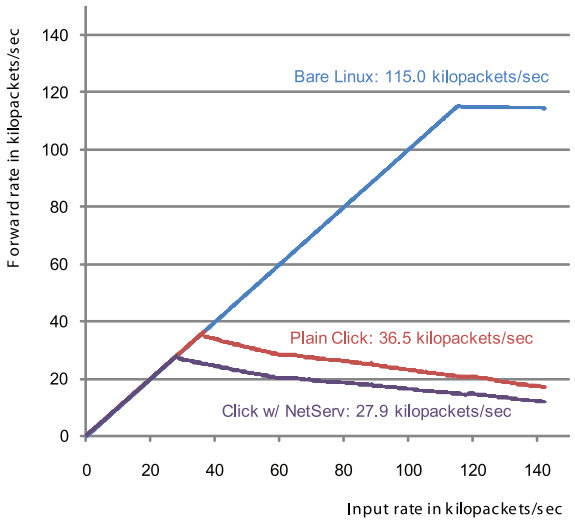

Figure 3: Comparison of the MLFFR of bare Linux, a user-mode Click router, and a user-mode Click router running NetServ, forwarding 64 byte packets.

Ethernet drivers, and Click version 1.7.0 rc1. Node 2 ran Ubuntu Linux 9.04, standard Ethernet drivers, and Click version 1.6.0.

4.2 Maximum Loss Free Forwarding Rate (MLFFR)

The overall test for MLFFR involves having node 1 send and count packets, node 3 receive and count packets, and node 2 forward packets between node 1 and 3. We compare the counts from nodes 1 and 3 to determine when and how many packets are dropped. We identify MLFFR as the highest packet rate for which the packet count at node 1 is the same as the packet count at node 3. Our measurements show that the MLFFR for Bare Linux is 115,000 packets/sec, Plain Click is 36,500 packets/sec, and Click with NetServ is 27,900 packets/sec as depicted in Figure 3.

For each test run, node 1 generates packets using the Click element *FastUDPSource* and counts them as they are sent. Node 3 simply discards them after they have been counted. We have node 1 run successive tests using every rate between 100 and 143,000 packets/sec in increments of 100. Each rate is run for 20 seconds. The tests are run with both 64 and 1500 byte packets. We also have both node 1 and 3 log the packet count every second while the test is running. Node 2 is run in each of three different scenarios for forwarding: 1) Bare Linux, 2) Plain Click, and 3) Click with NetServ. MLFFR is calculated using the data collected from the kernel logging.

Bare Linux forwarding is enabled by changing the value of `/proc/sys/net/ipv4/ip_forward` to 1. Node 2 uses the forwarding capability built into Linux kernel to handle packet forwarding.

Plain Click is a router configuration generated using the make-ip-conf.pl included with Click. This script creates a fully IP-compliant router. The configuration generated by this script will handle all typical forwarding duties expected of an IP router, including ARP requests and replies.

Click with NetServ uses the same configuration but inserts the NetServ element. The NetServ element is placed early in the path the packet travels: directly after it has been identified as an IP packet. The packet will travel through all the components depicted in Figure 2 with a single NetServ App Bundle installed. This bundle does nothing to the packet and is introduced to only gauge the overhead of sending a packet through NetServ.

Our tests for 64 byte packets show that there is sizable performance overhead when comparing Bare Linux to Plain Click. This was expected because node 2 is running Click in user-mode in the latter. This results in a kernel-to-user transition as the packet comes in to Click and then a user-to-kernel transition as Click forwards the packet out to the intended destination. These transitions result in penalties to the performance of the system. When we compare Plain Click to Click with NetServ, we see additional overhead introduced by the NetServ components. However, we see that the overhead of introducing NetServ on top of Click is significantly less compared to the overhead between Bare Linux and Plain Click.

When looking at the 1500 byte packet test, Bare Linux, Plain Click, and Click with NetServ are all capable of forwarding rates that are comparable: just over 8200 packets/sec. Each router levels off at the same maximum rate because they reach the bandwidth limit of our setup. This result is favorable, as real world use of NetServ is more likely to involve manipulating larger packets instead of sending minimum sized packets as fast as possible.

Since the MLFFR of 115,000 packets/sec for bare Linux was sufficiently close to the theoretical MLFFR of 148,800 packets/sec for 100 Mbps Ethernet connection [19], we tested to make sure that the number represented the limit of the bare Linux routing performance rather than the line limit. We replaced node 2 with a 10/100Mbps Ethernet switch. A direct connection between the two nodes causes the connection to run at 1Gbps. The switch forced a 100Mbps connection. This ensures comparability with our other test cases which use 100Mbps connections. Performing our MLFFR test in this scenario resulted in a rate of about 142,200 packets/sec. In order to ensure that this is the line limit for 100Mbps and not some other limit, we also directly connected node 1 and 3 allowing the connection to run at 1Gbps. This resulted in a forwarding rate that approached 500,000 packets/sec. These checks demonstrate that we are reaching the limit of Bare Linux and not another barrier.

5. RELATED WORK

Our work is fundamentally different from the active networking proposals such as ANTS [22], JanOS [20], NetScript [24] and Switchware [16]. In contrast to active networking, NetServ provides for virtualized services on current, passive networks by installing modules on the router control plane. Service invocation is signaling driven, not packet driven.

A service-centric view of the network core is not new. Tilman Wolf proposes a new abstraction for information transfer in the next generation Internet [23]. NetServ complements the idea, as it can provide the technology platform on which to implement the abstraction.

Much work has been done on virtualizing different parts of the Internet architecture. Their focus is sharing network resources such as bandwidth. NetServ's focus is providing a uniform hosting architecture for network services.

The DaVinci project [18] presents the design of a system that allows one physical network to support multiple classes of traffic. Major commercial routing hardware vendors, such as Cisco and Juniper, are also offering increasingly fine-grained network virtualization services for their customers [2, 14, 9]. Vyatta, an open-source routing platform vendor, also offers similar networking virtualization services [15].

The OpenSolaris Crossbow [12] project aims to enable network virtualization and resource control for each service or protocol such as HTTP or FTP. It does so by virtualizing the protocol stack and the NIC for each service.

The VROOM router project [21] presents "virtual routers" that can be moved from one physical node to another and controlled using network primitives. Egi *et al.* [17] evaluate implementation issues while designing a virtual router using the Xen virtual machine framework.

There are several other architectures that decouple control from the data plane in routers. The OpenFlow Switch [11] aims to allow a standard interface to routers to enable researchers to run experimental protocols on their campus networks. Ethane [5] provides a management model that aims to allow simple management and security in enterprise networks.

6. CONCLUSION AND FUTURE WORK

We described a prototype implementation of NetServ, an extensible architecture for core network services for the next generation Internet. We augmented the Click router with the Java-based OSGi framework to provide security, portability, resource control, and the ability to dynamically deploy service modules. Our measurement results indicate that the performance difference between our prototype and the Click router–essentially the processing overhead of the Java layer– is much smaller than the difference between the Click router and the Linux kernel. The difference between Click and Linux comes from the fact that the Click router runs as a user process and thus every packet incurs a transition from kernel to user mode. In a system where a module can be installed dynamically, such transitions are likely unavoidable, because running a service module inside a kernel would pose an unacceptable security risk.

As future work, we plan to enhance the NetServ framework with a full suite of security and resource control mechanisms. We also plan to develop a number of applications that can demonstrate the power of NetServ.

7. ACKNOWLEDGMENTS

The NetServ project is funded by the National Science Foundation under grant NSF-CNS #0831912 as a part of its Future Internet Design (FIND) initiative, and also by DOCOMO Communications Laboratories Europe.

8. REFERENCES

[1] Apache Felix. `http://felix.apache.org/`.

[2] Cisco network virtualization. `http://bit.ly/abknD`.

[3] The Click modular router project. `http://read.cs.ucla.edu/click/`.

[4] Eclipse Equinox. `http://www.eclipse.org/equinox/`.

[5] Ethane. `http://yuba.stanford.edu/ethane/`.

[6] JAR file specification. `http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html`.

[7] Java 2 platform security architecture. `http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html`.

[8] Java Native Interface specification. `http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html`.

[9] Juniper Networks Partner Solution Development Platform. `http://www.juniper.net/us/en/products-services/nos/junos/psdp/`.

[10] The NetServ project. `http://www.cs.columbia.edu/irt/project/netserv/`.

[11] The OpenFlow switch. `http://www.openflowswitch.org/`.

[12] OpenSolaris project: Crossbow: Network virtualization and resource control. `http://opensolaris.org/os/project/crossbow/`.

[13] OSGi Alliance. `http://www.osgi.org/`.

[14] Service-oriented network architecture. `http://www.cisco.com/en/US/netsol/ns629/networking_solutions_packages_list.html`.

[15] Vyatta network virtualization. `http://www.vyatta.com/products/virtualized.php`.

[16] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare Active Network Architecture. *IEEE Network*, May 1998.

[17] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley. Evaluating Xen for Router Virtualization. In *Computer Communications and Networks (ICCCN)*, pages 1256–1261, 2007.

[18] J. He, R. Zhang-shen, Y. Li, C. yen Lee, J. Rexford, and M. Chiang. DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In *Proceedings of CoNEXT*, 2008.

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[20] P. Tullmann, M. Hibler, and J. Lepreau. Janos: A Java-oriented OS for Active Network Nodes. *IEEE Journal on Selected Areas in Communications*, 19:501–510, 2001.

[21] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 231–242, New York, NY, USA, 2008. ACM.

[22] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH*, 1998.

[23] T. Wolf. Service-centric end-to-end abstractions in next-generation networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 79–86, Arlington, VA, 2006.

[24] Y. Yemini and S. D. Silva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 1996.