Toward a Practical Reconfigurable Router: A Software Component Development Approach

Ke Xu, Wenlong Chen, Chuang Lin, Mingwei Xu, Dongchao Ma, and Yi Qu

Abstract

Modern routers are no longer simple packet routing equipment. They are extremely complex systems that incorporate various network management functionalities. Due to system complexity, most commercial routers are developed by a few equipment vendors in a closed development pattern, which not only hinders the broad participation of most routing function research and development (R&D) teams, but also deter the wide deployment of novel network architectures such as Software Defined Networks (SDNs). In this article a practical approach is introduced to build an open, flexible, and modularized reconfigurable router. A reconfigurable routing software platform supporting functional modules is dynamically assembled, replaced, and updated in the form of components. So far the platform has been adopted by vendors such as Ruijie Networks in their commercial routers, and it is running well all the time. Moreover, a component development environment is provided, which consists of a code editor, a multi-platform compiler, and an automatic testing tool. A component sharing website has also been built to encourage the participation of various R&D teams and promote the spread of our open reconfigurable development pattern.

Routers are core network equipment that forward data packets between computer networks. Besides the fundamental packet routing capability, modern routers incorporate a variety of extended functionalities such as traffic management, packet filtering, and virtual private networks (VPNs). As a result, router systems are becoming extremely complex and creating exceedingly high barriers to network innovations.

Currently, most commercial routers are developed by a few giant vendors such as Cisco Systems, Inc. and Juniper Networks, and the design details are kept secret for commercial reasons. Routers following traditional closed development patterns are static, inflexible, and non-modularized, which causes several problems:

- Functional modules from different vendors cannot support interoperability.
- Specific functional modules cannot be dynamically installed, replaced, or updated.
- Numerous R&D teams and individuals are excluded from routing software component development.
- The closed development pattern and non-modularized structure leads to a tremendous amount of repetitive coding.

Ke Xu, Chuang Lin, Mingwei Xu, and Yi Qu are at Tsinghua University.

Wenlong Chen (corresponding author) is at Capital Normal University.

Dongchao Ma is with North China University of Technology.

In addition, the Internet has become one of the most important infrastructures in our daily life. The unimaginable complexity of the Internet and the economic tangles among different network participants including Internet Service Providers (ISPs) and Content Providers (CPs) are causing the Internet to become more ossified. We all recognize that network programmability is important to new protocol experiments and architectural innovations such as Software Defined Networks (SDNs) [1] and Named Data Networking (NDN) [2], and thus is critical to the future of the Internet. However, network programmability will never be achieved until network equipment programmability becomes a reality. In this article we introduce a practical approach to building an open, flexible, and modularized reconfigurable router that can support modular assembly and reuse in different functional levels, as well as component updates in the running state. This is recognized as an efficient way to realize router programmability and flexibility.

Two main technologies that are closely related to our approach are Click [3] and XORP [4]. Click is a flexible and configurable routing software architecture that combines simple elements into a system to achieve desired functions. Compared to our approach, Click only focuses on functional expansion of the router's data-forward plane and does not provide a component development environment. XORP is an open network platform that solves the code and API closure problem in routing software architecture. It implements IPv4/IPv6 protocols and provides a unified platform to configure them. XORP mainly concentrates on the control plane. Compared to Click and XORP, our reconfigurable routing

0890-8044/14/$25.00 © 2014 IEEE

software platform (reconfigurable platform for simplicity) includes all functional components in both the data-forward plane and the control plane. All approaches achieve high flexibility and easy configuration. However, at present only our reconfigurable platforms are adopted in commercial environments.

The contributions of this article are as follows. First we implemented a reconfigurable platform that includes a Virtual Operating System (VOS) platform layer that shields operating system differences and provides the upper layers with unified application interfaces (APIs), a component control layer that ensures the collaboration of components, and a routing service layer that actually performs routing functions.

The reconfigurable platform is component-based, and more components mean a shorter development cycle and more comprehensive routing capabilities. Hence, attracting more institutions into component development is critical. To facilitate the development process and ensure the quality and compatibility of components, we provided a component development environment and a unified standard. We also built a website to encourage the sharing of components. The development environment, standard, and website are building blocks of our open reconfigurable development pattern, which jointly form our second contribution.

So far the reconfigurable platform has been adopted by various vendors such as Ruijie Networks, Tsinghua Bitway Networking Technology Co., Ltd., and H3C Technologies Co., Limited. The platform is running reliably in their commercial routers. Based on the reconfigurable platform, we have completed several experiments such as virtual network and source address validation [5–7]. These practical uses and experiments have validated the correctness and effectiveness of our reconfigurable platform and development pattern.

The rest of this article is organized as follows. The next section presents the requirements and goals of our approach. We then introduce our reconfigurable platform. We illustrate the component development process, and then describe the component development environment. We then conclude our open reconfigurable development pattern. The last section concludes the article.

Requirements and Goals

We first clarify two important concepts that are frequently used in this article: component and meta-component. A component is a complete functional module in routing systems. It has a detailed function description and clear external interfaces and usually exists as a task or process. Typical components include Open Shortest Path First (OSPF), Boarder Gateway Protocol (BGP), TELNET, and so on. A meta-component is a complete data processing or functional step in a component with high cohesion. Compared to components, the code sizes of meta-components are smaller. Typical meta-components of routing systems are binary tree management and Hash storage of routing tables. Based on meta-components, code reuse can be obtained during component development. A meta-component can be reused multiple times by various components or a single component.

According to the above requirements, the goals of our approach are as follows:

- Reconfigurable platform. The reconfigurable platform should run effectively on different operating systems, and be adopted by various router vendors. In addition, the component collaboration and interaction needs to be properly handled.
- Component model. A component model should be provided, such that the features and behaviors including component management methods, component states and parameter management, component description and identification, can be specified clearly. The most important feature of the component is external interfaces, which shows the component's external characteristics and interaction capabilities. As to external interfaces, a unified standard is needed to ensure component compatibility.
- Component development environment. In such an environment, developers should be able to edit and compile components based on meta-components, and reconfigurable equipment can get online component installation and update. Typical routing operating systems such as VxWorks and Linux need to be supported by the environment.
- Component reliability. Component quality may vary since many institutions are involved in the development process. As a result, the quality and compatibility of components need to be ensured by methods such as automatic test tools and third-party evaluation authorities.
- Component sharing. Qualified components should be shared among different institutions in a simple and convenient way.

Reconfigurable Routing Software Platform

The architecture of a reconfigurable router is illustrated in Fig. 1a, and the main constituent of the reconfigurable platform (i.e. components) are developed following the process shown in Fig. 1b. A reconfigurable router can be divided into three main parts: hardware, operating system, and a reconfigurable platform. In this section we focus on the reconfigurable platform (component development is described later), which consists of three layers: the virtual operating system (VOS) platform layer, the component control layer, and the routing service layer.

Hardware Layer: There is no difference between reconfigurable and traditional router hardware layers. Both kinds of equipment contain an embedded processor, static random access memory (SRAM), dynamic random access memory (DRAM), a field programmable gate array (FPGA), and dedicated chips such as ternary content addressable memory (TCAM).

VOS Platform Layer: The reconfigurable platform may be adopted by various vendors, such that the router hardware and operating systems are different and sometimes incompatible. For example, typical router operating systems are VxWorks and Linux and the basic functions of these systems are essentially similar. However, their implementations, interfaces, and the methods of managing hardware resources are quite different. Therefore we need a middleware (i.e. VOS platform layer) to shield the differences below and provide unified APIs for layers above. At present, our VOS platform layer supports two typical router operating systems: VxWorks and Linux.

Component Control Layer: As shown in Fig. 1, the component control layer achieves component assembly and ensures proper collaborations between components. Component state management monitors components' running state and provides information for component control. Component parameter configuration configures core parameters, such that components can be adjusted to different operational requirements.

Component Library and Meta-Component Library: A component library provides components for routing software, while a meta-component library provides the basis for component development. It is worth mentioning that neither of them exists in a running router.

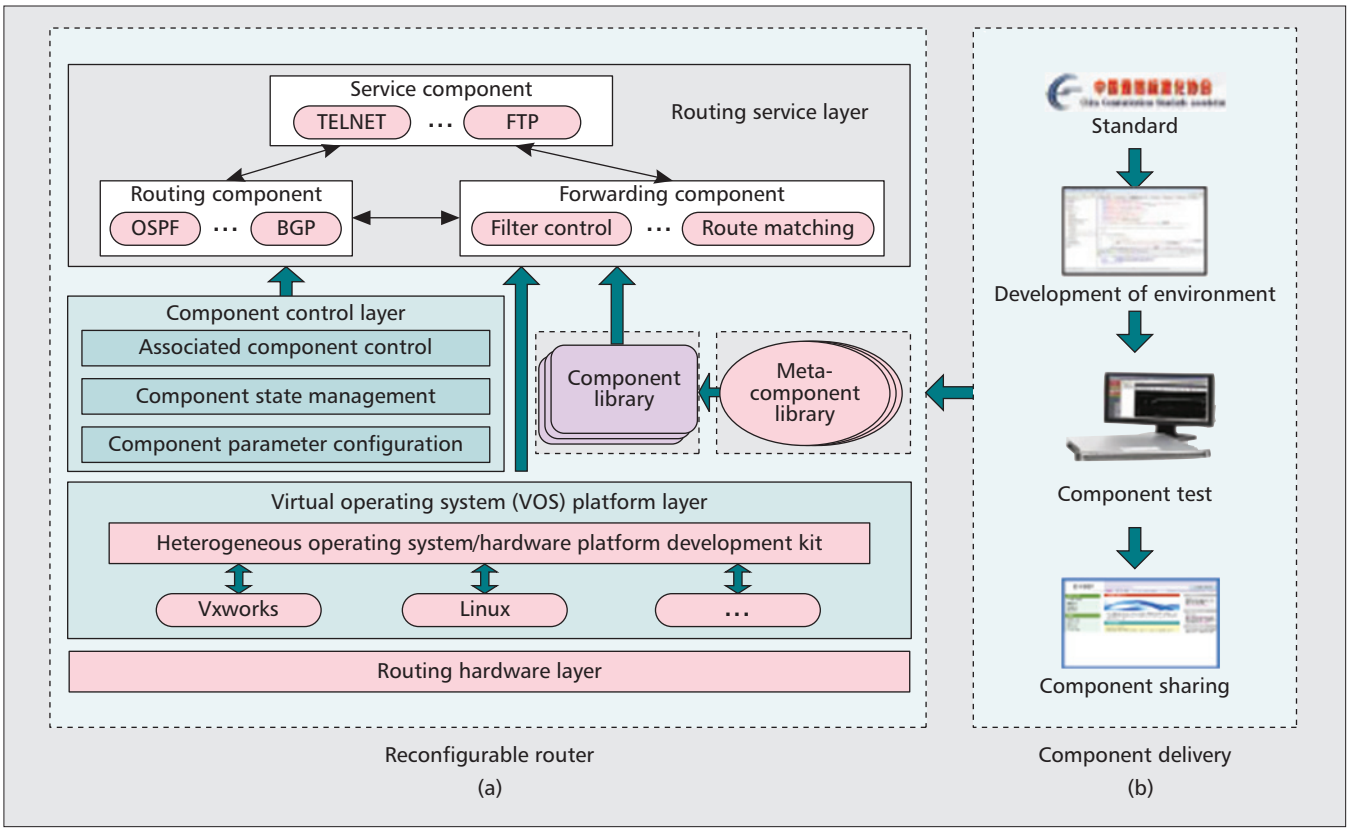


Figure 1. (a) Reconfigurable router and (b) component development.

Routing Service Layer: All routing operations (data forwarding, packet filtering, etc.) are accomplished on this layer through the collaboration of lower-layer components. Controlled by the component control layer, components exchange information through communication interfaces and work together to achieve route learning, packet forwarding, message processing, and other services.

Component Model

The core constituent of the reconfigurable platform is the component, and the most important aspect of the reconfigurable platform is component control, which includes the component model, interactions, and collaborations. All of these are discussed later in this article.

A component model describes the main characteristics and associated behaviors of components, defined as: $M_{component}$ = <CT, CI, SM, PC, ID, DI, OB>. A detailed description is as follows.

Control Topology (CT): Control components in running state, which mainly include component registration, cancellation, existence notification, and close notification.

Communication Interface (CI): In terms of control plane components, communication interfaces mainly include protocol message reception/transmission and common event message communication. As to data plane components, communication interfaces mainly include data message input/output.

State Monitoring (SM): The reconfigurable platform needs to monitor the running state of components and conduct controls based on this information. Component state management focuses on the process properties of components, such as the states of running, hang, and wait. We also pay close attention to statistical information such as the number of messages sent and received, and the sizes of routing tables on data plane components.

Parameter Configuration (PC): Component designers set important parameters (e.g. stack sizes, priorities, and storage sizes) to meet various application environments and router configurations. Parameters are configured statically before components are enabled.

Identifier (ID): The identifier indicates the destinations and sources of components during communication, e.g. the common message communication mechanism uses task ID as identifiers, while in socket communication IP addresses and port numbers are used as identifiers.

Description (DI): The description includes two parts: a description of communication interfaces in XML language, and component functions by a descriptive language given by component developers. For typical components developed based on Requests for Comments (RFCs), only RFC numbers are needed. Otherwise, a detailed description is required.

Object (OB): On the reconfigurable platform, all components exist in the form of compiled object files. This helps protect the intellectual property rights of component developers.

Component Interfaces and Associations

Management Interfaces of Components — A component agent (hereafter referred to as "agent") is responsible for registration, cancellation, and correlation of components. Component management interfaces are specifically designed for component management and are used to interact with the agent. Component management interfaces mainly include two output interfaces and two input interfaces with fixed names and usages (shown later in Fig. 2a):

- Component registration message M_IF_1: Once a component starts up, it sends a registration message (including component name, identifier, etc.) to the agent and reports its presence immediately.
- Component cancellation message M_IF_2: Normally, components send cancellation message to an agent before operation is stopped.
- Registration notification message M_IF_3: Sent to components by the agent. If component A applies for registration,

the agent will send registration notification messages to all components correlated with A. At the same time the agent will notify component A with information about all components correlated with it.

- Cancellation notification message M_IF_4: Sent to components by the agent. If component A applies for cancellation, the agent will send cancellation notification messages to all components correlated with A. Even if A does not send a component cancellation message, the agent will trigger a cancellation notification message after determining component A is stopped.

Communication Interfaces of Components — Besides management interfaces whose types and quantities are fixed, components have several input/output communication interfaces (C_IF_i in Fig. 2a) to interact with other components. Communication interfaces can be divided into two types: a protocol packet communication interface, which is designed to interact with other components for protocol messages, and a control message communication interface, which exists in all components.

The quantity and function of communication interfaces depends on specific requirements. For example, OSPF component interfaces mainly consist of protocol packet input/output, interface state reception, command configuration, and routing information generation. For a particular component, according to the directions of message transmission, communication interfaces can be divided into message input interfaces, which receive messages from upstream components, and message output interfaces, which send messages to downstream components.

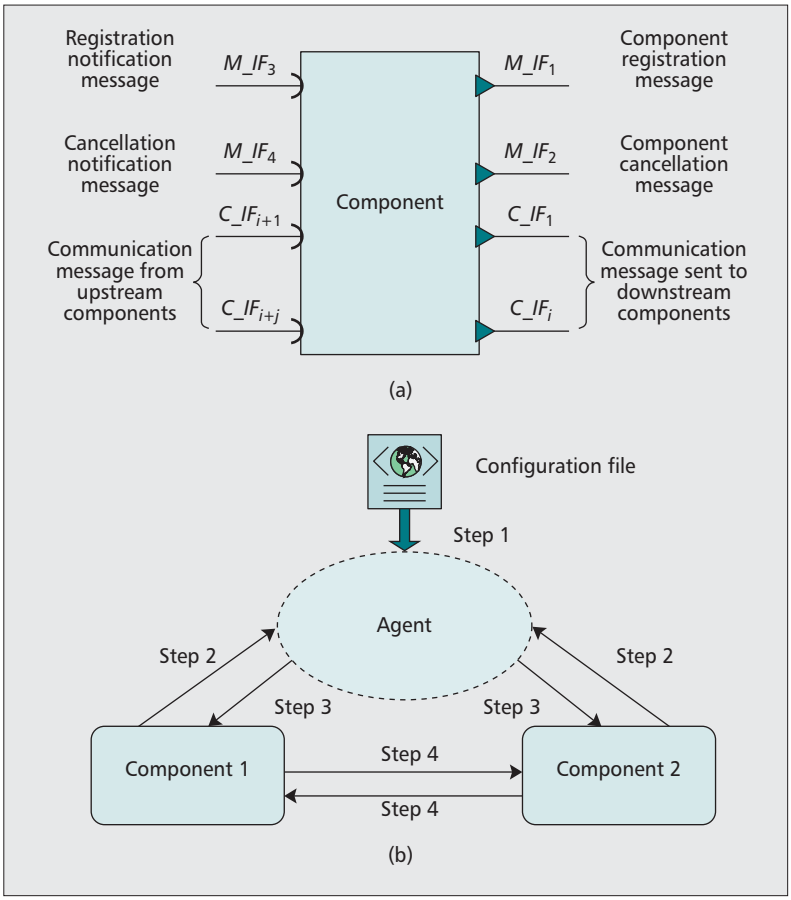


Figure 2. Interfaces and assembly of components.

Component Associations — Figure 2b describes an example of component association, where component registration/cancellation is implemented in the form of starting/stopping a process.

Step 1: System initialization. The agent reads the configuration files, which describe the component relationships.

Step 2: Components 1 and 2 are uploaded and then registered to an agent. The agent determines the relationships between components 1 and 2 based on the configuration file and their registration information.

Step 3: The agent forwards the communication ID of registered components to their upstream components.

Step 4: Components 1 and 2 send messages to each other through a communication ID obtained from the agent.

After component registration, the agent monitors components through heartbeat signals. When a component stops operation, the agent is responsible for notifying its upstream components.

Component Development Process

To ensure the quality and compatibility of components, we provide a unified component/meta-component development standard. It covers all stages of the component/meta-component development process including interface design, source coding, component/meta-component description, and automatic test. Accomplished components/meta-components are submitted to third-party authorities to evaluate their compatibility with other components/meta-components that interact with them. Only those components/meta-components that are compliant with the development standard would go through the third-party evaluation and be put into the component store.

Key Stages

The component development process includes the following key stages (as shown in Fig. 3):

1. **Component design.** Components are designed based on analysis of modules with high coherence in routing systems. All functions and interfaces of meta-components should be generally applicable. Besides, high storage/processing performance are also required.
2. **Component description.** A large number of components may be developed by various institutions. Component descriptions can help developers understand functions and features of components, and contribute to the management of component libraries and indexes. In addition, it is also the basis of component tests.
3. **Component development.** Developers design specific components to meet the functional requirements under component descriptions and the unified development standard. Components could be developed based on existing meta-components to accelerate the development process and enhance the code reuse rate. Accomplished components are submitted to a general component library.
4. **Component test.** Components in the general component library should be tested by the automatic testing tool (described in detail in the section on automatic testing) to ensure consistency between component functions and descriptions as well as the normalization and effectiveness of a component's external interfaces. The certified components will be collected into a certified component library.

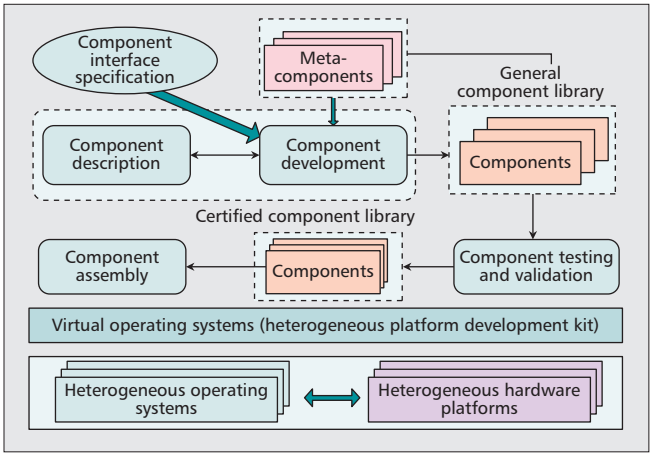


Figure 3. Key stages of component development process.

5. **Component usage.** Routing software developers can select components from the certified component library. ISPs can also select components from the certified component library to carry out dynamic function installation or replacement.

Component/Meta-Component Classification Principle

As mentioned before, a component is a complete functional module, while a meta-component is a complete data processing or functional step. A well-defined principle that guides the partition of routing system into components and meta-components does improve the development of the systems and promote the component/meta-component reuse rate. However, the boundary between components and meta-components is sometimes ambiguous. To deal with this problem, all of the commonly used components/meta-components are recorded and described in detail. Developers propose applications whenever a new type of component/meta-component is developed. Only after comprehensive discussion and approval can the new type of component/meta-component be submitted into the component/meta-component store.

Automatic Test

Before submitting to third-party authorities for interaction evaluation, components/meta-components should be tested by the automatic test tool, which is part of the open reconfigurable development pattern. The automatic test tool includes two parts: an automatic test management server (ATMS) and an automatic test execution server (ATES) (shown in Fig. 4). The ATMS is responsible for user interactions such as test parameter setting and test description. We run it on Windows because Windows provides a better user experience. The ATES runs on Linux since test execution involves a huge amount of computation, and high efficiency is required.

Utilizing the ATMS, all component/meta-component information such as input/output interfaces and important parameters are displayed to users after an XML parser extracts this information from the component/meta-component description file. Whenever test information is configured, the ATMS calls the ATES to execute the actual test process.

Based on the test information, the ATES automatically generates drivers and stubs. All of the drivers, stubs, and the to-be-tested components/meta-components are registered to the agent. After registration, drivers call the components/meta-components to be tested to verify its input interfaces and stubs are called by the to-be-tested components/meta-components to evaluate their output interfaces. All of the test results are compared with the component/meta-component description file to ensure consistency between description and the actual function.

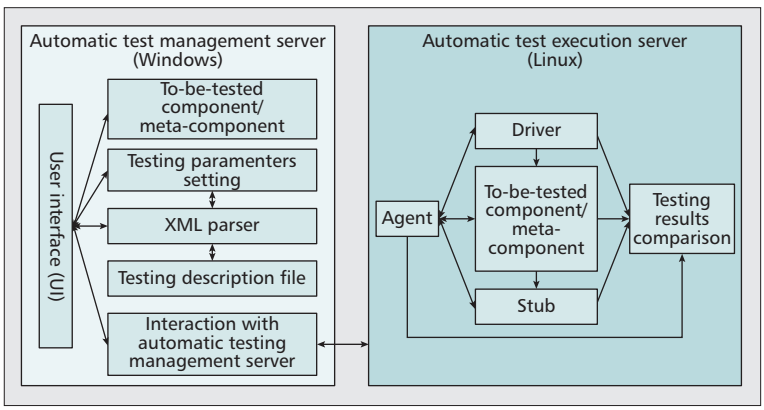


Figure 4. Automatic test.

Component Sharing

We have built a component store to accumulate high quality components. Before a component can be accepted by the store, a series of conformance and quality tests are performed by a third-party organization. At present, the third-party organization is the Research Institute of Telecommunication Transmission (RITT) of the Ministry of Posts and Telecommunications [8], a test authority in China. Conformance tests consist of two parts. One is performed to evaluate whether the to-be-tested component is in compliance with the component development standard. The other part is performed to assess its accordance with its own description file. Quality tests are also needed to ensure the function correctness and efficiency of the to-be-tested component. As described earlier, only those components in compliance with the component development standard and its own description file and with high quality would go through the third-party evaluation and be put into the component store.

We have also built a website [9] to share our component development standard, the library of accomplished components/meta-components, and the reconfigurable development environment. Our website also supports the upload/download of components/meta-components. A variety of institutions will share their resources through the website. In order to facilitate learning and development, the website also offers a detailed schematic introduction and user manuals. The widespread use of the reconfigurable platform and development environment will bring enormous social benefits and also encourage the healthy development of the Internet.

Component Development Environment

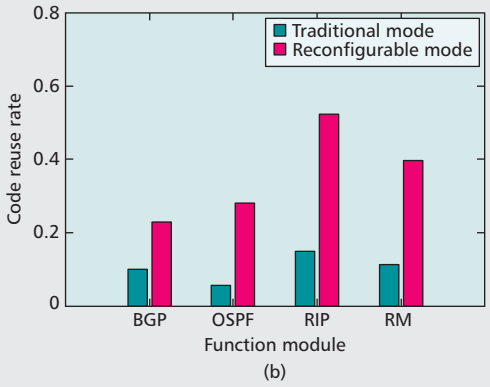
Development Environment

We developed a component development environment, which includes the following parts:

1. **Code editor.** Developers code directly on the development environment.
2. **Multi-platform compiler.** Supporting Linux, VxWorks (PowerPC) and VxWorks (x86).
3. **Component/meta-component development management.** Supporting meta-components referenced by components.

Meta-component	Function description	Components
OLL	One-way link list	All
DLL	Doubly linked list	All
HS	Hash storage	BGP, SAV, PT
WDOCM	Network development oriented memory management	All
FSM	Finite state machine	BGP, OSPF, RIP, SAV
DMC	Distributed messaging communication	All
TM	Timer management	All
SM	Skbuf management	PF, SAV, TEn/De, PT
NISM	Network interface status management	BGP, OSPF, RIP, RM PF, SA, TEn/De, PT

(a)



(b)

Figure 5. (a) Major meta-components; (b) code reuse rate comparisons.

4. **Component simulation.** Designing stubs and drivers to conduct component testing according to the component description.
5. **Component assembly control.** Forming specific routing functions through component assembly.

The development environment shields the differences of operating systems, making the entire development process based on a unified environment. The development environment is the foundation of our open reconfigurable development pattern and it supports all aspects of the reconfigurable platform. Moreover, it promotes the widespread use of the open reconfigurable development pattern by providing any individual or R&D team with a visual programming environment.

Practical Deployment

Under the unified development standard, our project team has developed 10 meta-components, as shown in Fig. 5a. (The third column lists the components using the corresponding meta-component). We also developed the following 12 components: BGP, OSPF, Routing Information Protocol (RIP), Routing Management (RM), Packet Forwarding (PF), Source Address Validation (SAV), Tunnel Encapsulation/De-encapsulation (TEn/De), Packet Translation (PT), TELNET, File Transfer Protocol (FTP), Secure Shell (SSH), and Management Information Base (MIB).

Some components (e.g. BGP, OSPF, and RIP) have already been used by multiple types of routers manufactured by Ruijie Networks, Tsinghua Bitway Networking Technology Co., Ltd., and H3C Technologies Co., Limited, etc. All components in commercial routers are performing reliably and have passed the function conformance test performed by RITT. We compared the code reuse rates between a traditional development pattern and our open reconfigurable development pattern. The results are shown in Fig. 5b. Our open component development pattern significantly improved the code reuse rate in the development of functional components such as BGP, OSPF, RIP, and RM. Moreover, the workload of source coding is also reduced significantly after we developed components based on universal meta-components.

Open Reconfigurable Development Pattern

The open reconfigurable development pattern for the reconfigurable platform is shown in Fig. 6. First, institutions (e.g. router vendors, R&D teams, and individuals) use the component development environment to develop their own components/meta-components. The development process is obliged to follow the component development standard. Then all components/meta-components are tested by automatic test tools and third-party authorities. The tests can be done through virtual simulation or integrating components/meta-components in a typical router system in actual equipment and network scenarios. The components/meta-components that passed the authority tests can be added to a component/meta-component store for sharing.

Institutions require components capable of extracting certified components from the component store. R&D teams or equipment manufacturers will leverage the certified components for their routing product development. In order to increase or update routing functions of network equipment, ISPs can utilize certified components for their reconfigurable routing equipment. Furthermore, networks can provide programmability through reconfigurable equipment.

The benefits of our open reconfigurable development pattern, which is similar to the Android market and the App Store (iOS), are twofold. On the developer side, they are motivated to develop various components/meta-components and put them into the store to obtain increased economic benefits. On the consumer side, the development process of

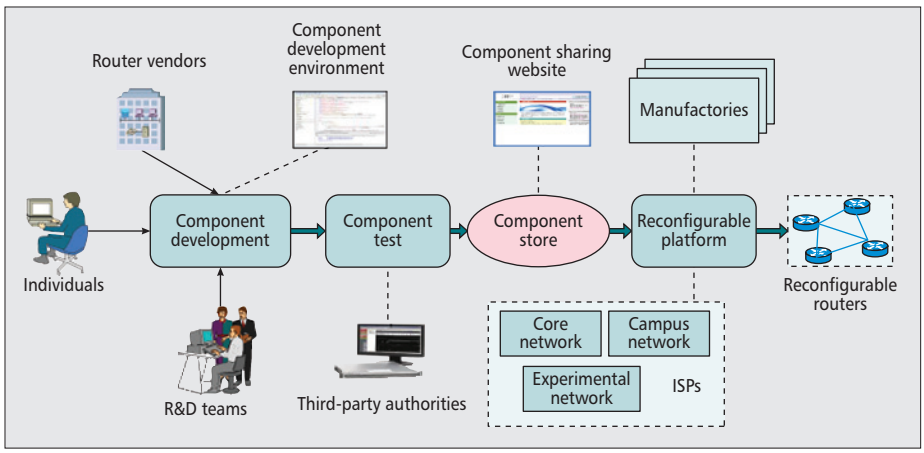


Figure 6. Open reconfigurable development pattern.

routing systems are accelerated and simplified utilizing the high quality components/meta-components in the store.

Conclusion

This article proposed a reconfigurable routing software platform and the corresponding open reconfigurable development pattern, which overcome the closed, non-component deficiencies of traditional routing systems. The platform allows software modules in routing systems to be dynamically assembled and replaced in the form of components. It also supports flexible component reuse, which shortens the development cycle of routing systems. The open reconfigurable development pattern supports component development of any R&D team that follows a unified standard. These tested components will be integrated into the store.

Our reconfigurable platform is one of the ways to provide router programmability, which is critical to SDN. To support SDN, during the deployment phase we could deploy reconfigurable routers and controllers in networks. Reconfigurable routers are responsible for packet forwarding. Whenever a new type of protocol needs to be deployed, corresponding components could be installed dynamically on reconfigurable routers under the command of controllers. In this way, network programmability and flexibility are ensured.

Our project team has completed the development of 10 meta-components and 12 components, and provided a component development environment, which simplified the development process. Moreover, we established a components/meta-components store, provided a website for component sharing, and implemented a reconfigurable platform. So far, the open reconfigurable platform has been adopted by various vendors such as Ruijie Networks, Tsinghua Bitway Networking Technology Co., Ltd., and H3C Technologies Co., Limited, in their commercial routers, and it is running reliably.

Acknowledgment

This research is supported by New Generation Broadband Wireless Mobile Communication Network of the National Science and Technology Major Projects (2012ZX03005001), NSFC Project (61170292, 61373161), 973 Program (2012CB315803), 863 Program (2013AA013302), and EU Marie Curie Actions Evans (PIRSES-GA-2010-269323).

References

[1] N. McKeown *et al.*, "Openflow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Commun. Review*, vol. 38, no. 2, Apr. 2008, pp. 69–74.
[2] L. Zhang *et al.*, "Named Data Networking (ndn) Project," Relatório Técnico NDN-0001, Xerox Palo Alto Research Center (PARC), 2010.
[3] E. Kohler *et al.*, "The Click Modular Router," *ACM Trans. Computer Systems*, vol. 18, no. 3, Aug. 2000, pp. 263–97.
[4] M. Handley, O. Hodson, and E. Kohler, "Xorp: An Open Platform for Network Research, *ACM SIGCOMM Computer Commun. Review*, vol. 33, no. 1, Jan. 2003, pp. 53–57.
[5] M. Xu *et al.*, "Veganet: A Virtualized Experimentation Platform for Production Networks with Connectivity Consistency," *IEEE Network*, vol. 26, no. 5, Sept. 2012, pp. 15–21.
[6] Y. Cui *et al.*, "4over6: Network Layer Virtualization for IPv4–IPv6 coexistence," *IEEE Network*, vol. 26, no. 5, Sept. 2012, pp. 44–48.
[7] G. Hu *et al.*, "A General Framework of Source Address Validation and Traceback for IPv4/IPv6 Transition Scenarios," *IEEE Network*, vol. 27, no. 6, Nov. 2013, pp. 66–73.
[8] http://www.ritt.cn
[9] http://www.netlab.edu.cn/drrp/index2.html

Biography

KE XU [M'02–SM'09] (xuke@tsinghua.edu.cn) received his Ph.D. from the Department of Computer Science and Technology, Tsinghua University, where he serves as a full professor. He has published more than 100 technical papers and holds 20 patents in the research areas of next generation Internet, P2P systems, Internet of Things (IoT), network virtualization and optimization. He is a member of ACM. He has guest edited several special issues in IEEE and Springer Journals. Currently he is holding a visiting professor position at the University of Essex.

WENLONG CHEN (wenlongchen@sina.com) received a Ph.D. degree. He is currently a lecturer in the College of Information Engineering of Capital Normal University. His research interests include network protocol and network architecture.

CHUANG LIN (chlin@tsinghua.edu.cn) is a professor in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is an Honorary Visiting Professor at the University of Bradford, UK. He received the Ph.D. degree in computer science from the Tsinghua University in 1994. His current research interests include computer networks, performance evaluation, network security analysis, and Petri net theory and its applications. He has published more than 400 papers in research journals and IEEE conference proceedings in these areas, and has published four books. Professor Lin is a senior member of the IEEE. He served as the General Chair of the ACM SIGCOMM Asia Workshop 2005 and the 2010 IEEE International Workshop on Quality of Service (IWQoS 2010). He is an associate editor of *IEEE Transactions on Vehicular Technology*, and an area editor for the *Journal of Computer Networks*.

MINGWEI XU (xmw@cernet.edu.cn) received the B.Sc. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1994 and 1998, respectively. He is a full professor in the Department of Computer Science and Technology, Tsinghua University. His research interests include computer network architecture, high speed router architecture, Internet routing, and network virtualization.

DONGCHAO MA (madongchao@csnet4.cs.tsinghua.edu.cn) has a Ph.D. and is an associate professor at the Information Engineering Institute, North China University of Technology. His research interests include Next Generation Internet, network management, and traffic flow management.

YI QU (quy11@mails.tsinghua.edu.cn) received his B.Eng. degree in software engineering from the University of Electronic Science and Technology of China in 2011. Currently he is a Ph.D. student in the Department of Computer Science and Technology of Tsinghua University. His research interests include wireless networks and wireless sensor networks.