中国科学技术大学

本科毕业论文

SDN 路由器的容错软件架构

作者姓名：　　李博杰

学科专业：　　计算机科学与技术

导师姓名：　　谭焜 研究员　华蓓 教授

完成时间：　　二〇一四年六月八日

University of Science and Technology of China

A dissertation for bachelor's degree

Fault-Tolerant Software Architecture for a SDN Router

Author's Name: Bojie Li

speciality: Computer Science

Supervisor: Dr. Kun Tan Prof. Bei Hua

Finished time: June 8th, 2014

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开　□保密（____年）

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘 要

路由器软件的任何一部分出错或升级时往往需要冷重启，造成分钟级别的网络中断。本文设计和实现了一个容错的路由器软件架构，包括路由协议和管理员构成的客户端、路由器数据库、解决客户端间路由规则冲突的同步服务和可编程路由器芯片 SDK。本路由器架构实现了路由协议之间的逻辑隔离，并在上述组件之任一出错或升级时不会中断数据平面，而且能够在该组件重启后可预测的时间内自动恢复控制平面。

关键词：路由器 交换机 容错 软件定义网络

ABSTRACT

Traditional router requires cold reboot if any component of the router software fails or needs to be upgraded. Network traffic is likely to be interrupted for minutes in the meantime. This work designs and implements a fault-tolerant software architecture composed of four components: clients including routing protocols and administrator, the router information base (MiniDB), a daemon to resolve rule conflicts among clients (SyncD) and the SDK for programmable switching chip. This architecture allows any component to fail or upgrade without interrupting data plane, and the control plane will automatically recover within predictable time after the component restarts.

Key Words: Router, Switch, Fault-tolerant, Software Defined Networking

目 录

第一章 引言

传统路由器的软硬件架构是相对封闭的，路由器操作系统的任何一部分出错或需要升级时，往往需要重启整个路由器，重启过程中路由器所连接的所有网络链路都会中断。在数据中心里，通常采用 Clos 拓扑结构 [1]，每台服务器只连接到一个机架顶上的交换机（Top-of-Rack switch），这台交换机一旦重启，机架内的所有服务器就会有长达几分钟的网络中断 [9]，这对服务的高可用性是很大的威胁。

现有关于路由器高可用性的研究 [5][6] 大多需要增加冗余链路和冗余设备，这会带来较高的成本。而路由器厂商提出的“Warm Reboot”解决方案只是缩短了操作系统重启的时间，路由器芯片仍然需要重新初始化，会带来几十秒的数据平面中断 [9]。

事实上，从微软 Azure 数据中心的运行经验看，路由器芯片发生故障的概率是很低的；路由器操作系统由于处于内网而没有外部安全威胁，很少需要升级。容易出错和需要升级的部分包括：管理平面（管理员使用的控制台 console 或 Web 界面）、路由协议、路由信息库、路由器芯片 SDK。这些组件的关系如图 1 所示。

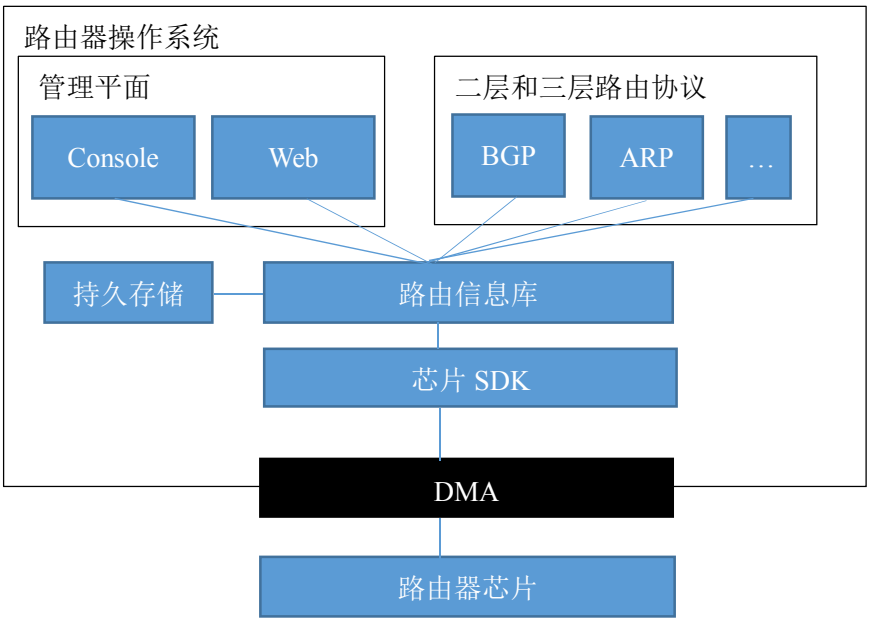

图 1 传统路由器软件架构

从表面上看起来，直接重启上述组件之任一不会影响路由器芯片的工作，然而现有商用路由器缺少故障恢复的机制，也没有为提供单独升级或重启某个组件开放接口。商用路由器的软件架构是封闭的，然而 ServerSwitch 实验平台 [2] 使

得定制路由器软件架构成为可能。本工作基于修改版的 ServerSwitch 实验平台，采用了有较高市场占有率的 Broadcom Trident II 交换机芯片 [3]，实现了容错的路由器软件架构。

我们从两个方面来实现路由器的高可用性。一方面，我们把不同路由协议从逻辑上隔离开来，由中心化的同步服务来解决路由协议间的配置冲突，减少逻辑错误；另一方面，我们把路由信息库的功能拆分为简单而稳定的轻量级路由信息库和相对复杂的同步服务，避免单点故障，如图 2 所示。

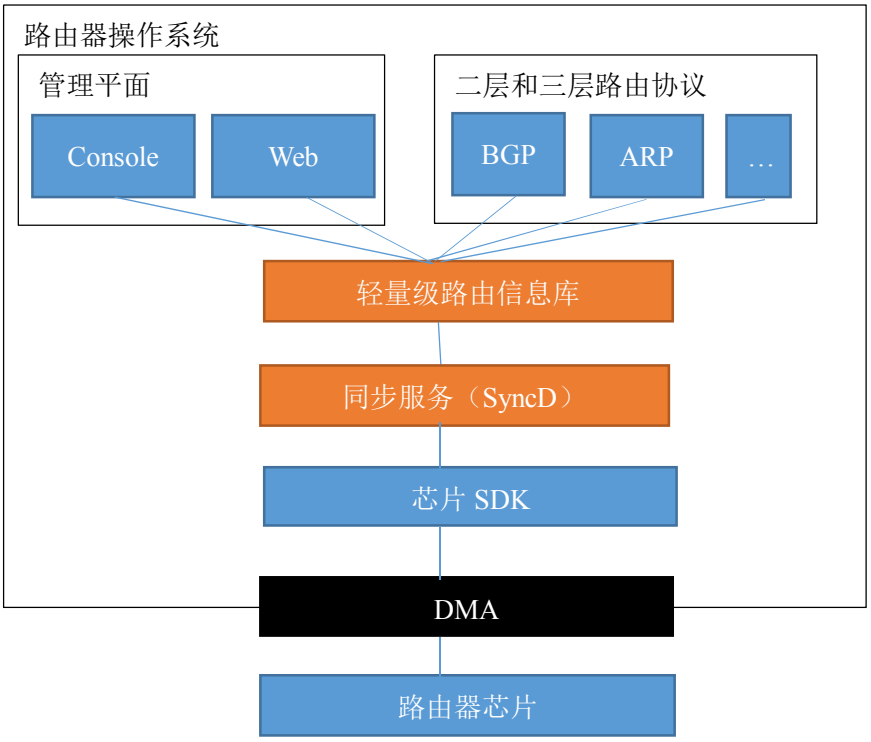

图 2 容错路由器软件架构

其中，轻量级路由信息库负责与管理平面和路由协议建立稳定的连接，提供方便的修改和查询接口，并检查配置的一致性；同步服务按照优先级解决各路由协议间的配置冲突，并适配芯片 SDK 的接口。各路由协议的配置在轻量级路由信息库和同步服务中各保存一份，降低了跨进程通信的开销，而且在两个组件之一出错或需要升级时可以从另一个组件恢复配置信息。

当各组件分别出错或需要升级时，对控制平面和数据平面的影响范围和服务中断时间如表 1：（详细分析见后）

出错或升级组件	控制平面	数据平面
某个管理组件	该组件，秒级	无

某个路由协议	该组件，秒级	无
轻量级路由信息库（MiniDB）	整个控制平面，秒级	无
同步服务（SyncD）	整个控制平面，10 秒级	无
芯片 SDK	整个控制平面，分钟级	无

表 1 单个组件出错或升级的影响

两个以上组件同时出错从概率上是几乎不可能的，但为了严谨性仍然需要讨论：（详细分析见后）

同时出错的组件	控制平面	数据平面
某个管理组件+其他	除该组件外，由其他出错组件决定	由其他出错组件决定
某个路由协议+其他	除该组件外，由其他出错组件决定	由其他出错组件决定
MiniDB 和芯片 SDK	整个控制平面，分钟级	无
SyncD 和芯片 SDK	整个控制平面，分钟级	无
MiniDB 和 SyncD	整个控制平面，可用性：10 秒级；一致性：路由协议全量更新后保证一致	在路由协议全量更新前的几十秒内，新增配置可能与原有配置冲突
MiniDB、SyncD 和芯片 SDK	整个控制平面，可用性：分钟级；一致性：路由协议全量更新后保证一致	在路由协议全量更新前的几十秒内，网络中断

表 2 两个及以上组件同时出错的影响

本文第二章介绍 ServerSwitch 实验平台；第三章描述同步服务（SyncD）的设计；第四章分析系统各组件出错或升级时系统的行为；第五章展示容错软件架构的测试结果；第六章总结。

第二章 背景

2.1 ServerSwitch 架构

ServerSwitch [1] 是一个基于可编程交换机芯片的实验平台。本文使用的修改版 ServerSwitch 是一块 PCIE 扩展卡，有 4 个 40 Gbps 外部以太网接口和 4 个 40 Gbps 内部以太网接口（以太网卡在板上，通过 PCIE 接入系统）。通过内部以太网卡，主机可以直接向交换机发数据包，或接收流过交换机的数据包；通过外部以太网接口，交换机与交换机之间、交换机与普通服务器之间可以连接起来构成更复杂的拓扑结构。

ServerSwitch 相比普通交换机的优势在于 ServerSwitch 运行在标准 x86 架构的计算机上，CPU 处理能力强大，而且有 4 个标准以太网接口而不是普通交换机的一个特殊 CPU 接口。首先，ServerSwitch 以太网接口的吞吐量和 CPU 的处理能力使得网络内部（in-network）的高速数据包处理成为可能；其次，标准以太网接口编程简单，既可以使用成熟的操作系统网络协议栈，又可以使用 NDIS 过滤驱动，而不需要通过专有的 SDK 从 CPU 接口收发数据包。

ServerSwitch 所用的 Broadcom Trident II 交换机芯片处理数据包的流程如图 3 所示。

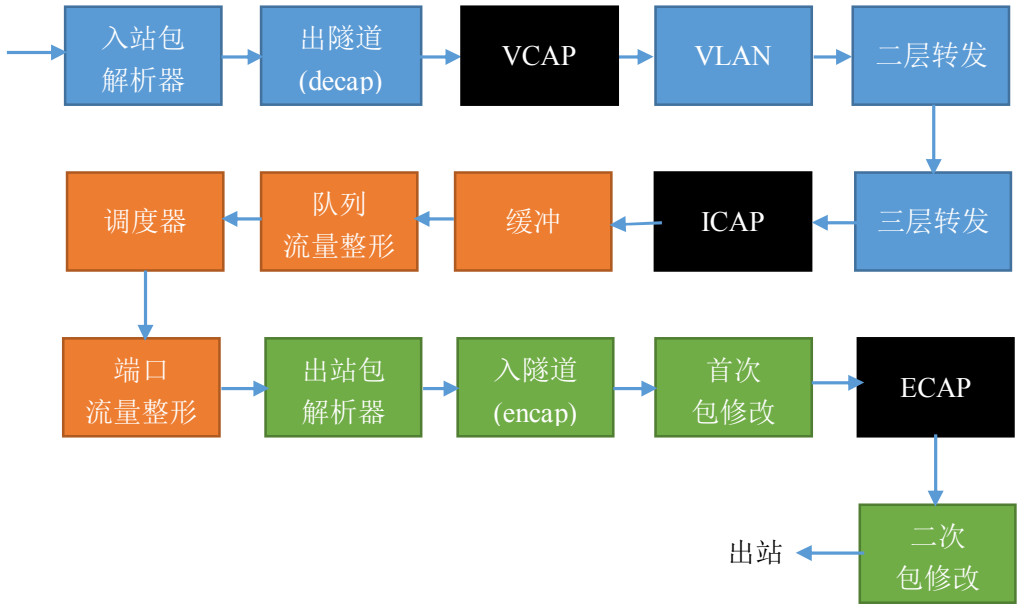

图 3 Broadcom Trident II 芯片数据包处理流程

从入站包解析器到 ICAP 的部分是入站流水线（Ingress Pipeline），这段流水线把数据包的内容放进片上内存，把解析出的数据包头信息和欲对数据包执行的操作放进入站队列。从缓冲到端口流量整形的部分是内存管理单元（MMU），这

段流水线是把入站队列里的包按照调度优先级和流量整形的规则，移动到出站队列里。从出站包解析器到二次包修改的部分是出站流水线（Egress Pipeline），这段流水线从出站队列依次取出数据包头和欲执行的修改操作，再从片上内存取出数据包的内容，完成数据包的重组。

ServerSwitch 所用的数据包缓冲区内存由 61440（60K）个单元构成，每个单元 208 字节，也就是总共可以缓冲 12.2 MB 的数据包。每个数据包至少占据一个缓冲区单元，第一个单元的前 64 字节用于存储解析后的数据包头，后面 144 字节用于存储数据包内容（包括原始数据包头）；如果数据包的长度大于 144 字节，则需要另外的缓冲区单元，每个附加的单元可以存储 208 字节的数据包内容。ServerSwitch 流水线中只传递指向数据包缓冲区的指针而不会传递数据包内容，也就是每个数据包在缓冲区内只存储一次。

2.2 硬件流水线中的表

ServerSwitch 硬件流水线中的大多数阶段都是“表”的形式，每张表分为匹配（match）部分和动作（action）部分。匹配部分可能匹配数据包头的一个特定偏移量，也可能匹配随流水线流动的内部标签，有精确匹配、直接索引查询、前缀匹配和模糊匹配四种模式；动作部分可能修改随流水线流动的内部标签，也可能修改数据包头的一个特定偏移量。

精确匹配在硬件中是用 hash 表和 RAM 实现的，采用拉链法解决冲突；硬件首先计算待匹配项的 hash 值，找到链表，再顺着链表逐个精确匹配。直接索引查询使用 RAM 实现，直接读取对应的内存地址。前缀匹配是用 hash 表、RAM 和硬件逻辑实现的。

所谓模糊匹配，就是匹配模式的每一位都可以是一个 0/1 值或者“不关心”。模糊匹配是用 TCAM（Ternary Content-Addressable Memory）实现的，TCAM 可以并行匹配所有表项，如果一个数据包同时匹配上多个表项，以表中最前面的项为准。只有 VCAP（VLAN CAP）、ICAP（Ingress CAP）、ECAP（Egress CAP）三张表可以执行模糊匹配。

流水线中“首次包修改”之前的表对数据包头的修改都在流水线末尾的“首次包修改”处执行，也就是大部分表匹配的是都是原始数据包头。ECAP 对数据包头的修改在“二次包修改”处执行，也就是 ECAP 匹配的是修改后的包。

流水线中不同的表可能有着相同类型的动作，后面的动作会覆盖前面的动作，也就是后面的表优先级更高。

ServerSwitch 所使用的 Broadcom Trident II 芯片拥有 800 多张表，为简明起

见我们只描述常用的一小部分表中常用的匹配项和动作，并忽略了硬件表中的很多细节。

表名	匹配	动作
出隧道（Decap）	精确：NVGRE Tenant ID	Decap 修改 Internal VLAN ID
Port VLAN	精确：入站端口	设置内部 VLAN ID 设置内部优先级 可选设置二层隧道
Packet VLAN	精确：802.11Q 外层 VLAN ID	设置内部 VLAN ID 设置内部优先级 可选设置二层隧道
VCAP	模糊	见后
目的 NAT	精确：目的 IP	设置新目的 IP
二层/三层选择	精确：内部 VLAN ID、目的 MAC 地址	决定是否经过三层逻辑
二层交换	精确：内部 VLAN ID、目的 MAC 地址	设置出站端口、VLAN ID
三层直接路由	精确：目的 IP	设置下一跳（nexthop）或 ECMP 组 可选设置内部优先级
三层前缀路由	前缀：目的 IP	设置下一跳或 ECMP 组 可选设置内部优先级
ICAP	模糊	见后
ECMP 组	精确：ECMP 组，数据包头 hash 值	设置下一跳
三层下一跳（nexthop）	直接索引：下一跳	设置出站端口 设置目的 MAC 地址 可选设置三层隧道
三层网络接口	直接索引：网络接口	设置源 MAC 地址
优先级映射	直接索引：内部优先级	设置严格优先队列
严格优先队列	直接索引：严格优先队列	使用严格优先级进行调度
源 NAT	精确：源 IP	设置新源 IP
三层隧道	直接索引：三层隧道	设置外层数据包头

二层隧道	直接索引：二层隧道	设置外层数据包头
ECAP	模糊	见后

表 3　常用的硬件表说明

VCAP、ICAP、ECAP 均可模糊匹配数据包头的各标准字段（包括源 MAC、目的 MAC、外层 VLAN ID、内层 VLAN ID、源 IP、目的 IP、IP 优先级、四层协议、四层源端口号、四层目的端口号、DSCP 标记等）和一些数据包处理流水线上的内部状态（如入站端口、内部 VLAN ID、内部优先级、三层网络接口），改变数据包头的各标准字段和/或流水线内部状态，和/或执行丢包、镜像、限速、入隧道等特殊动作。也就是说，能在其他表里完成的操作基本上都可以在 TCAM 中完成。VCAP、ICAP、ECAP 每张表可以匹配的字段和可以执行的操作都不相同，VCAP 和 ECAP 的功能是相对受限的。

由于 TCAM 的成本比较高，VCAP、ICAP、ECAP 的表项都是相对受限的，分别有 1K、4K、1K 个表项。其中最强大的 ICAP 由 12 个分片构成，前 4 个分片各有 512 个表项，后 8 个分片各有 256 个表项。之所以需要分片，是因为需要模糊匹配的表项并不是相同宽度的，例如只匹配 IPv4 五元组（源 IP、目的 IP、四层协议、四层源端口号、四层目的端口号）的表项与匹配 IPv6 五元组与源 MAC 地址、目的 MAC 地址的表项宽度显然明显不同，如果 TCAM 按照最大可能的宽度设计，那么很多空间会被浪费。

Broadcom Trident II 中的 ICAP 所用的 TCAM 宽度为 275+106 位，其中 275 位是匹配数据包头的标准字段，106 位是匹配入站端口的位图（此芯片最多支持 106 个入站端口）。如果欲匹配的表项比 TCAM 宽，就需要把两个甚至多个分片拼接起来，构成匹配宽度更大的 TCAM。

2.3　路由器上的典型应用

本文所描述的容错路由器软件架构主要应用于机架顶上的交换机（Top-of-Rack Switch）。

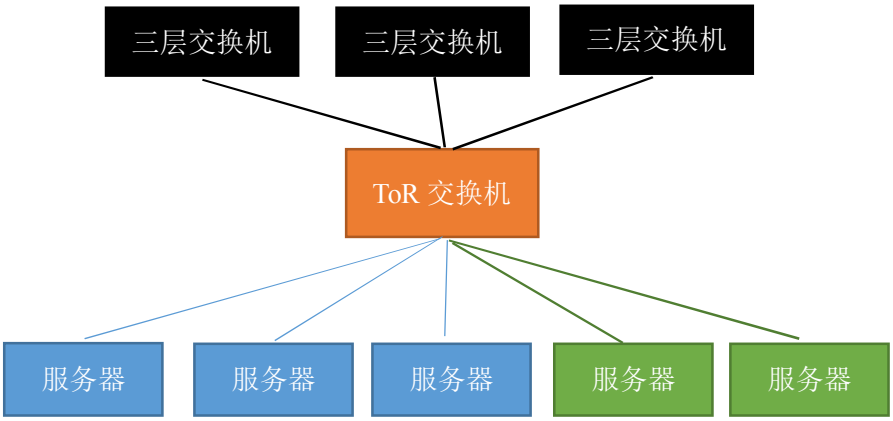

图 4　ToR Switch 拓扑结构

在典型的网络中心拓扑结构中，ToR 交换机一方面与多个三层交换机（Aggregation Switch）相连接，另一方面与机架内的服务器相连接，每台服务器接到 ToR 上的一个网络接口。机架内的服务器划分为若干个 VLAN。ToR 的职责主要包括：

1. 两台同一机架、同一 VLAN 下的服务器，要能够通过 ToR 直接通信；
2. 访问其他机架的服务器时，数据包要进入三层网络，要根据 VLAN ID 打上不同的 NVGRE/VXLAN 数据包头（encapsulation），并使用 ECMP 在多条三层链路间进行负载均衡；
3. 从三层网络发到 ToR 的数据包要解开 NVGRE/VXLAN 数据包头（decapsulation），根据隧道包头中的客户 ID 信息，发到相应 VLAN 的服务器。
4. 三层网络内的路由使用 BGP/OSPF 协议。
5. 回复服务器或三层网络发来的 ARP 请求。
6. 自动学习服务器的二层 MAC 地址。
7. 管理员要能配置流量控制、流量优先级和防火墙策略。
8. 通过 SNMP 协议监控链路状态和网络流量。

根据以上 ToR 的职责，可以把 ToR 上的应用划分为几类：

应用类型	涉及的表
VLAN 与隧道控制（SDN）	VLAN, Tunnel Encap, Tunnel Decap
数据中心三层网络控制（SDN）	三层网络接口（L3 Interface），三层下一跳（L3 Nexthop），三层直接路由（L3 Direct Routing），ECMP
BGP/OSPF	三层网络接口，三层下一跳，三层前缀路由（L3 LPM

	Routing）
ARP	二层交换（L2 Switching）
MAC 地址学习	二层交换（L2 Switching）
流量控制、防火墙	VCAP，ICAP，ECAP
SNMP	VCAP、ICAP、ECAP、物理接口等表中的计数器

表 4　ToR Switch 上的典型应用

上述应用存在不同应用管理同一张表的情况，每个应用所管理的多张表还存在引用依赖（referential dependency），例如三层前缀路由表的每个表项都会引用三层网络接口表和三层下一跳表。不同应用之间必然存在冲突表项的优先级问题和删除表项时的引用计数问题。现有路由器的处理逻辑是 ad-hoc 的，路由器软件的编写者稍不注意就可能出错。

我们的容错软件架构为每个应用提供一组虚拟表，每个应用只能看到自己添加的表项，这些数据也原样保存在轻量级路由信息库（MiniDB）中，由同步服务（SyncD）对不同应用的表项按照预先设置的优先级解决冲突，并维护引用计数，以防误删除其他应用仍在使用的表项或在路由器内留下垃圾表项。

表之间的优先顺序由路由器应用之间的优先顺序决定。系统管理员需要在路由器应用接入 MiniDB 时指定其优先级（任意两个应用的优先级不能相同）。例如在前面的例子中，一个合理的优先顺序是流量控制和防火墙、VLAN 与隧道控制、数据中心三层网络控制、BGP/OSPF、ARP、MAC 地址学习、SNMP。

第三章 同步服务（SyncD）设计

本文描述的路由器容错软件架构由客户端、轻量级路由信息库（MiniDB）、同步服务（SyncD）和 Broadcom 交换机芯片 SDK 构成。其中客户端是包括各种路由协议在内的路由器应用，站在 MiniDB 和 SyncD 的角度看，它们就是客户端。这是一个合作项目，本人负责的主要是同步服务（SyncD）的设计与开发。

同步服务（SyncD）通过 WCF（Windows Communication Foundation）框架从轻量级路由信息库（MiniDB）读取表结构、客户端间优先级和各客户端的表内容，用冲突解决算法得到合并后的表内容，再以远程过程调用（RPC）的方式更新到芯片 SDK。

MiniDB 采用增量方式向 SyncD 插入表项，每个表项的状态可能是生效、部分生效或未生效，会被写回 MiniDB 以便客户端查询。表项没有生效的原因可能是与更高优先级的表项冲突、硬件表已满、芯片 SDK 无响应等。表项部分生效只可能在前缀匹配表中出现，原因是与更高优先级的表项部分冲突，例如在三层前缀路由表中，在插入一条较高优先级的“10.0.0.0/16 走 2 号端口”之后，较低优先级的“10.0.0.0/8 走 1 号端口”就会成为部分生效状态。由于在模糊匹配表内检查冲突的时间复杂度较高（与表项数目成线性关系），模糊匹配表不检查冲突，也就是即使有冲突也会被插入硬件表并显示为生效状态，但低优先级的冲突表项事实上没有效果。

为保证客户端之间的隔离性，并确保发生故障后所恢复内容的一致性，SyncD 中的冲突解决算法要保证：只要每个客户端得到的最终表相同，不论客户端采用了何种插入、删除的操作序列，不论客户端之间的操作顺序，冲突解决后各客户端表项的状态和硬件表的内容也应该等价。

3.1 表的抽象定义

轻量级路由信息库（MiniDB）是连接路由协议客户端与同步服务的纽带，提供基于表的通用信息存储和一组添加、查询、修改、删除表项的原语（API）。同步服务需要遵循 MiniDB 对表结构的定义。

与关系数据库中的表类似，每张表由若干预先定义的列构成，列之间是没有顺序的。表中的条目，或称“行”，就是给每列一个确定的取值（包括 NULL）。

我们将表分为两类：有序表和无序表。有序表中的条目是有顺序的，顺序代表优先级，模糊匹配表属于有序表。无序表中的条目是没有顺序的，也就是表中的条目构成一个集合，精确匹配表、直接索引表和前缀匹配表都属于无序表。按

照这里的定义，关系数据库中的表都是无序表，因此我们不能直接采用关系数据库作为存储。

每张表都必须有且仅有一个主键（primary key），主键是若干列的集合。表中不允许存在两个主键相同（即主键所包含的每列的值都相同）的条目。一般来说，主键就是所有匹配字段的集合，例如三层直接路由表的主键是目的 IP，二层转发表的主键是内部 VLAN ID 和目的 MAC 地址，直接索引查询表的主键就是索引。

表之间可以有引用依赖（referential dependency），即表 A 的某一列必须从表 B 的某一列中已经存在的值中取值。为简化系统设计，我们规定，引用依赖必须指向直接索引查询表的主键，两张表之间至多有一个引用依赖。这样引用依赖可以省略列名，直接说表 A 引用依赖表 B。例如三层直接路由表引用依赖三层下一跳表，三层下一跳表引用依赖三层网络接口表，三层直接路由表引用依赖 ECMP 组表。

直接索引查询表的主键从逻辑上说是可以任意分配的，只是在物理上必须是从 0 开始的索引。在保持引用依赖的前提下，客户端可以任意设置直接索引查询表的主键，同步服务会把各客户端分配的重复主键进行合并，再映射到物理上从 0 开始的索引。所谓重复主键，就是除了主键以外所有列都相同的条目，对直接索引查询表来说这样的条目显然是可以合并的。

3.2 冲突表项的定义

我们首先给出各种类型表中“冲突”的定义，详细解释在“增量修改”一节。

对所有类型的表，同一客户端内不可能出现冲突的表项。MiniDB 会检查每张表的一致性，如果发现冲突则会拒绝插入。

精确匹配表是有主键的。如果出现两个不完全相同的条目，但主键完全相同，则认为它们发生了冲突。如果出现两个完全相同的条目，不认为它们发生了冲突，但如果这样的两个条目在同一客户端内，MiniDB 会把它们自动合并，也就是 SyncD 获取到的表中不会出现来自同一客户端的两个完全相同的条目。

直接索引表的所有条目都没有冲突。

前缀匹配表中的主键一定包括前缀匹配部分。如果两个不完全相同条目的主键完全相同，则认为它们发生了冲突。两个完全相同的条目不认为是冲突的（MiniDB 的自动合并仍然适用）。两个条目如果主键中仅前缀匹配部分不同，且它们的前缀匹配部分有重叠（一个是另一个的子集），则认为它们发生了冲突。

模糊匹配表中，两个不完全相同条目的主键如果完全相同，则认为是冲突的。两个完全相同的条目不认为是冲突的（MiniDB 不会对它们做自动合并）。两个条

目的主键匹配规则如果有重叠，也就是存在一组匹配项的取值能够同时匹配两个条目，则认为它们是冲突的。

3.3 从客户端表生成硬件表

为了保证同步服务重启后能够根据各客户端的表重新生成一致的硬件表，我们需要定义一种全量生成硬件表的算法作为标准。

首先定义各客户端表中所有条目的优先级全序关系：

- 不同客户端间条目的优先级顺序取决于客户端间的优先级顺序；
- 同一客户端内条目的优先级顺序：
  - 模糊匹配表按照表内的顺序；
  - 精确匹配表、直接索引表按照条目被插入的时间顺序（从优先级的意义上讲，插入总是在“末尾”）；
  - 前缀索引表前缀长度较长的优先级较高，前缀长度相同的按照条目被插入的时间顺序。

下面描述全量生成硬件表的算法：

1. 硬件表初始状态为空。
2. 将所有客户端表内的条目按照优先级从高到低放入临时表，所有表项都标记为未生效。
3. 从临时表内依次取出条目：
   a) 如果硬件表已满，则退出生成过程；
   b) 如果表的类型是模糊匹配，不论是否有冲突，都插入到硬件，并标记为已生效；
   c) 如果与已经插入硬件的所有条目都没有冲突，则将其插入到硬件，并标记为已生效；
   d) 如果表的类型是前缀匹配，且已经插入硬件的与之冲突的条目的前缀匹配部分都是它的前缀匹配部分的真子集，则将这个条目插入到硬件，并标记为部分生效；
   e) 不然，跳过这个条目，保持其未生效状态。

上述生成算法能保证生成的硬件表具有下列性质：

1. 表内无冲突：
   a) 除前缀匹配表外，任何两个冲突的条目不能被同时插入硬件；
   b) 对前缀匹配表，不存在两个来自不同客户端的、冲突的、被同时插入硬件的条目，其中优先级较低的匹配部分是优先级较高的子集；

2. 对所有未被插入硬件的条目，不存在与之冲突、比它优先级低但被插入了硬件的条目；
3. 如果一个没有被插入硬件的条目的所有冲突条目都比它优先级低，则不存在比它优先级低但被插入了硬件的条目。
4. 如果硬件表未满，不存在一个未被插入硬件的条目，能够被插入硬件而不违反以上三条性质。

3.4 增量修改

3.4.1 表的等价性定义

由于客户端对表的修改是增量的，为了提高效率，我们必须支持表的增量修改，即把客户端对表的增量翻译到硬件表的增量，同时保证增量生成的硬件表与全量生成的硬件表等价。

我们首先给出表“等价”的定义：

- 对精确匹配表和前缀匹配表，两张表等价当且仅当两张表的主键构成的两个集合相同，且主键相同的条目也完全相同；
- 对直接索引表，两张表等价当且仅当两张表的主键（除索引外的所有列构成主键）构成的两个集合相同；（等价性与索引无关）
- 对模糊匹配表，两张表等价当且仅当两张表的条目数量相同，且位置相同的条目也完全相同。（模糊匹配表是有序表）

可以证明，所有满足 3.3 节所述四条性质的硬件表都是等价的。因此，我们只需要证明下列增量生成算法生成的硬件表始终满足这四条性质。

3.4.2 精确匹配表的增量修改

在插入一个新条目时，如果出现两个不完全相同的条目，但主键完全相同，则两个条目中优先级较低的一个会被置为未生效状态。如果出现两个完全相同的条目，且没有与它们冲突且比它们优先级都高的第三个条目，则这两个条目都会被置为生效状态。

例如，ARP 协议设置网关 MAC 为 00:01:02:03:04:05，于是在二层转发表中插入一行（VLAN=1，MAC=00:01:02:03:04:05，Port=L3Forward），其主键为（VLAN，MAC）。接下来，10 号口上一台恶意的服务器设置自己的 MAC 为 00:01:02:03:04:05，并往交换机上发包，MAC 地址学习功能学习到一条二层转发规则（VLAN=1，MAC=00:01:02:03:04:05，Port=10）。这条规则与 ARP 协议所设

置的二层转发规则的主键相同，是冲突的。又由于 ARP 协议的优先级高于 MAC 地址学习功能的优先级，（Port=10）的恶意规则会被置为未生效状态，不会对二层转发构成影响。

使用 hash 表来实现主键冲突的检测，以主键作为 hash key，采用拉链法解决 hash 冲突。将来自不同客户端的完全相同的条目组成“条目组”，每个条目组由主键 hash 值、条目内容、生效状态、客户端列表（如果没有其他客户端有完全相同的条目，则此列表只有一项）、条目组优先级（由优先级最高的条目决定）。主键相同的条目组按照优先级从高到低的顺序排成一个链表，只有链表头的条目组被插入硬件表。

为了在硬件表满的情况下用优先级高的条目替换优先级低的条目，需要维护一个未被插入硬件表的主键构成的大根堆和一个已被插入硬件表的主键构成的小根堆，排序规则是主键链表头部的条目组之优先级。小根堆的根应该始终大于大根堆的根，否则就会违反性质（3）。显然，当硬件表不满的情况下大根堆应该是空的。

插入新条目时，首先看是否有其他客户端的完全相同条目，如果是则加入条目组（如果其优先级是条目组中最高的，则将条目组在主键相同者之链表中移动到合适的位置）；不然，则新建一个条目组，看有没有主键相同的条目组，如果有则按优先级插入链表，如果没有则新建一个链表插入 hash 表。

对新建主键的情况，如果硬件表未满，就将其插入硬件。否则比较新主键的优先级与小根堆之根的优先级，如果其优先级较低，则插入大根堆；如果其优先级较高，则将小根堆的根移到大根堆、从硬件表中移除，并将新主键插入小根堆和硬件表。

删除条目基本上是插入的逆过程。首先从条目组中删除条目，并调整条目组的优先级。如果条目组由于优先级的改变，从主键相同的条目组链表中从头部移开了，需要从硬件中删除该条目组，并添加现在处于链表头部的条目组。如果条目组中仅有一个条目，则此条目组被删除，若其在主键相同的条目组链表中处于头部的位置，且删除后链表不为空，则在硬件中插入当前处于链表头部的条目组；若链表为空，删除这个主键。

对删除主键的情况，如果删除前硬件表为满，则从大根堆中找到根（优先级最高的主键），将其插入硬件表，并从大根堆移动到小根堆。

3.4.3 直接索引表的增量修改

直接索引表的索引是可以任意分配的，但在客户端内部必须保持一致。直接索引表的条目之间没有“冲突”的概念，因为它们在芯片流水线中并不匹配任何

数据包头或内部状态，只是被其他表所引用。直接索引表的条目处于“未生效”状态的可能原因只有硬件表已满、芯片 SDK 无响应。

未插入硬件表的小根堆与已插入硬件表的大根堆与精确匹配表类似，不再赘述。

直接索引表除索引外的所有列构成主键，计算 hash 值并存入 hash 表，采用拉链法解决 hash 冲突。将来自不同客户端的主键相同的条目组成“条目组”，每个条目组由主键 hash 值、主键、生效状态、物理索引、客户端列表构成。为了从客户端索引查询硬件表索引，每个客户端还需要一张 hash 表把客户端索引映射到同步服务分配的硬件表索引。

插入新条目时，如果主键已经存在，则将该客户端加入客户端列表。如果主键不存在，则生成一个新的条目组，如果硬件表未满，分配最小的空闲物理索引（使用位图来维护物理索引的分配情况）；如果硬件表已满，则将其插入大根堆。

3.4.4 前缀匹配表的增量修改

前缀匹配表中，冲突关系形成一棵树。每个节点表示一个条目，两个节点之间的连线表示它们之间存在冲突，靠近树根的条目“覆盖”了靠近树叶的条目，也就是靠近树叶的条目匹配长度更长。

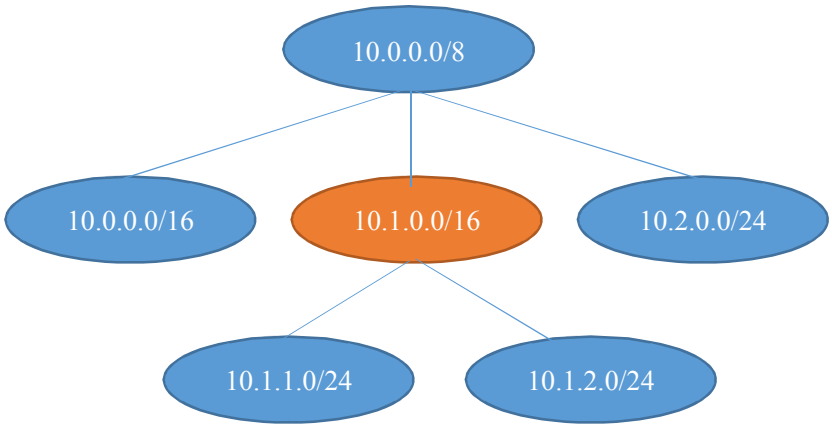

图 5 前缀匹配表的冲突关系示意

设欲插入的条目是 A，如果 A 在树中的任何一个祖先比它的客户端优先级高，则根据性质（1b），A 不能插入。

否则试着插入 A，把 A 标记为已生效，不论硬件表是否已满。按照条目优先级从高到低的顺序（条目优先级是全序关系，而客户端优先级可能相同），考虑 A 在树中的子孙：

- 已经被插入的子孙 C：

- 如果 C 比 A 的客户端优先级高，不变，并把 A 标记为部分生效的；
- 如果 C 与 A 来自相同客户端，不变；
- 如果 C 比 A 的客户端优先级低，删除 C。

- 尚未插入的子孙 C：
  - 如果 C 比 A 的客户端优先级高，则 C 比 A 的所有祖上优先级高，早就应该被插入，这种情况不应该出现；
  - 如果 C 与 A 来自相同客户端，且此时硬件表未满，插入 C；
  - 如果 C 比 A 的客户端优先级低，不变。

上述操作之后，有可能超过了硬件表条目数量的限制（至多一条）。如果超过了，需要删除一个条目，则选择删除所有已插入条目中优先级最低的，并更新其子孙条目（如果有的话）的生效状态。

上述操作之后，有可能硬件表未满，如果存在未插入条目，则按照未插入条目的优先级自高到低查找，如果一个条目的所有来自不同客户端之祖先比它的优先级低（即满足性质 1b），则它可以插入。重复本过程直到硬件表满或未插入条目为空。

确定要插入和删除硬件表的条目之后，作为一个批处理操作提交给芯片 SDK 执行，以降低硬件表内条目处于不一致状态的时间。

数据结构方面，需要以优先级为排序依据，维护已插入条目的小根堆，维护未插入条目的二叉平衡排序树，以便快速找到已插入条目中优先级最低的一个条目、未插入条目中优先级最高的若干个条目。需要维护条目间冲突关系构成的树。

3.4.5 模糊匹配表的增量修改

模糊匹配表中采用不检测冲突、只考虑优先级的插入方法，是考虑到模糊匹配表检测冲突比较耗时：用逐个匹配的朴素算法，增量更新时检测冲突所需时间与表项数量成正比，全量更新时检测冲突所需时间与表项数量的平方成正比。

由于所有条目按照优先级构成一个线性表，如果整个线性表能够容纳在硬件表中，则全部条目都可以插入；不然，硬件表中的条目就是线性表中的前 N 个条目，其中 N 是硬件表的大小。

插入或删除一个条目时，如果不在前 N 个条目的范围内，则无需对硬件表做任何修改；如果在前 N 个条目范围内添加一个条目到 M 位置，则需要在硬件表中将 M 到 N-1 位置的条目依次后移，再把新条目插入 M 位置；如果在从 N 个条目范围内的 M 位置删除一个条目，则需要在硬件表内将 M+1 到 N 位置的条目依次前移，再把原来 N+1 位置的条目（如果有的话）插入硬件表的 N 位置。在模糊匹配的硬件表内，相同的两个条目中靠后的条目是不会被任何数据包匹配

上的，因此在硬件表条目的移动过程中，硬件表始终是处于一致的状态。

3.5 插入表项到硬件

一个表项在通过冲突检测，决定要被插入硬件后，还有最后两件事要做：把客户端分配的索引号转换成硬件表中的索引号，并调用芯片 SDK 提供的 API。

如果表 A 引用依赖表 B（定义参见 3.1 节），则 B 是一个直接索引表，根据 3.4.3 节，客户端分配的索引未必是硬件表中的索引。3.4.3 节中从客户端索引到硬件表索引的映射 hash 表就在此时派上用场，这一步的动作就是把客户端索引替换成硬件表索引。

由于不同硬件表在芯片 SDK 中的 API 形式不同，我们只好为每个硬件表写一个适配类，将同步服务的插入、删除操作映射到芯片 SDK 中的相应 API 接口。

第四章 出错与升级情况分析

4.1 系统看门狗

系统看门狗（watchdog）监控路由器各组件的运行。每个组件是操作系统里的一个后台进程。当某个组件进程退出时，看门狗能得知这个事件，并立即启动该进程的一个实例。

当管理员希望升级某个组件时，需要把新的组件文件放在指定的目录，然后向看门狗进程发消息。看门狗进程收到消息后会首先检查新组件文件的完整性，杀死该组件的后台进程，用新组件文件覆盖掉原来的组件文件，然后启动该进程的一个实例。

组件的出错或升级都可以看成组件进程的重新启动，因此放在一起讨论。

4.2 单个组件出错或升级

4.2.1 管理平面的某个组件

当管理平面的某个组件出错或需要升级时，重启这个组件后，它可以从轻量级路由信息库（MiniDB）中读取表的内容。

4.2.2 路由协议

当某个二层或三层路由协议组件出错或需要升级时，重启这个组件后，它会按照路由协议获取最新的路由信息，并全量推送到 MiniDB。

例如，对分布式路由协议 BGP，其路由信息是依据与相邻节点的链路信息交换得到的，当 BGP 协议启动后，只需从相邻节点分别获取链路信息，再运行算法，即可得到完整的路由表，这也是标准 BGP 客户端启动时所做的事。

再如，对二层 MAC 地址学习功能，其 MAC 地址与物理端口的对应关系是根据从端口发来的数据包学习得到的。在该组件重启之后，只要某个物理端口发来了数据包，就能学到其 MAC 地址。物理机器一般每隔几秒就会发数据包，因此这个学习过程应该是很快的。在最坏情况下，假设该物理机器从不发包，则正常的 MAC 地址学习也会在几分钟（一个可配置的超时）后忘掉这个 MAC 地址，也就是 MAC 学习组件重启后的几分钟后能保证与该组件不重启时得到的（MAC，端口）映射信息相同。

既然这些路由协议组件有自我恢复能力，为什么现有路由器不能做到重启时不中断数据平面呢？这是因为现有路由器升级一个组件时，就会重新启动操作系统并重新初始化芯片，从芯片重新初始化到路由协议自我恢复的时间里，芯片内的配置是不完整的，因此会造成数据平面中断。而我们对组件的升级做到了细粒度的控制，实现一个组件升级时其他组件正常运行，也就是从来不会有重新初始化芯片的情况。

路由协议重启后主要可能存在的问题是重启前存在但重启后应当被废弃的规则。例如 MAC 地址学习组件重启前有一条（MAC，端口）映射，重启后这个 MAC 地址在物理上离开了这个端口，则这条映射不会被 MAC 地址学习组件知道，但 MiniDB 中仍然残留着这样一条垃圾配置。然而路由协议一般有全量同步的机制，每隔一段时间（一般是几十秒）会把自己的所有配置提交给 MiniDB，我们对这种全量同步采用“flush”接口而不是普通的“insert”接口，也就是全量同步之后 MiniDB 与路由协议内的状态是一致的。对于没有全量同步机制的路由协议，我们为其增加周期性的全量同步，即根据其内部状态生成完整的表并提交给 MiniDB。

4.2.3 轻量级路由信息库（MiniDB）

轻量级路由信息库（MiniDB）是简单而稳定的，基本不需要升级。如果它出错或需要升级，在 MiniDB 重启后，会首先从外部配置文件读取表结构信息，然后从 SyncD 读取重启前各客户端表的内容及生效状态，读取完毕后建立索引。MiniDB 恢复完成前拒绝客户端（管理平面和路由协议）的连接，因此不会出现数据不一致问题。MiniDB 重启造成控制平面的中断时间主要是从 SyncD 读取信息和建立索引的时间。

4.2.4 同步服务（SyncD）

同步服务（SyncD）如果出错或需要升级，重启后会从 MiniDB 读取各客户端的表内容，按照优先级解决冲突并生成最终的表，将最终的表通过芯片 SDK 接口写入硬件，再将表中各条目的生效状态写回 MiniDB。为保证一致性，在恢复完成前，SyncD 不接受 MiniDB 修改表中条目的请求。由于解决冲突算法和写入硬件需要较长的时间，SyncD 恢复需要 10 秒量级的时间（视路由器内配置条目数量而定）。

4.2.5 芯片 SDK

芯片 SDK 是路由器芯片厂商提供的，是系统中最复杂的部分。我们把它作为一个独立进程来运行，并通过远程过程调用（RPC）来调用其 API。

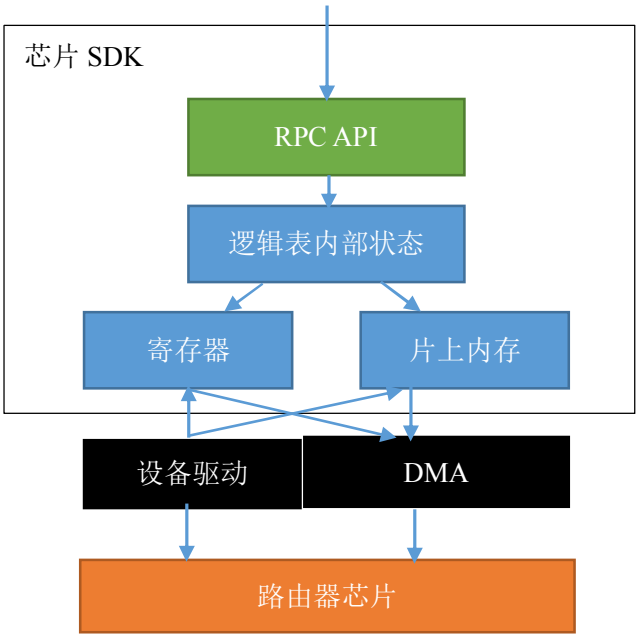

图 6 芯片 SDK 内部结构

路由器芯片与路由器操作系统的交互通过 PCIE 总线 DMA（Direct Memory Access）的寄存器和片上内存来进行。寄存器主要是一些相对零散的配置和芯片状态，而片上内存包括各种“表”和收发数据包的缓冲区。芯片 SDK 是一个用户态进程，在初始化时从内核态设备驱动申请寄存器和片上内存的 DMA 内存映射，之后 SDK 就不再访问内核态设备驱动了，而是直接通过 DMA 来修改芯片配置和读取芯片状态。

由于路由器芯片非常复杂，而且不同型号芯片的寄存器和片上内存定义有很多不同，芯片厂商已经做了大量的工作，提供了接口相对统一的 SDK，使得开发者可以比较方便地编程。相当于芯片厂商已经抽象出了若干逻辑表，并做好了逻辑表和硬件表之间的映射。SDK 内部维持着逻辑表的状态，这个状态是由硬件表唯一确定的，但由于读取寄存器和片上内存的开销较大，SDK 把状态缓存到了本地内存中。我们的工作完全基于 SDK 所提供的 API，不会直接访问芯片的寄存器和片上内存。

如果 SDK 需要升级，首先需要关闭 SDK 进程。SDK 退出后并不会影响数据平面，只会造成控制平面中断。系统看门狗通过给 SDK 传递命令行参数的方式告诉它这是一次重启而非全新启动。SDK 首先初始化逻辑表（内部数据结构），从寄存器和片上内存读取当前配置，重新生成逻辑表的状态，然后开始接受外部

调用；SyncD 连接上 SDK 后就会将其内部状态全量同步到 SDK，SDK 计算出新的逻辑表状态并增量更新寄存器和片上内存。在这个恢复过程中，全量读取片上内存需要一分钟左右，后面的步骤大约是 10 秒的量级，因此 SDK 出错或升级会造成一分钟左右的控制平面中断。由于芯片不会被复位或初始化，而且在恢复的最后一步，寄存器和片上内存是增量更新的，因此不会造成数据平面的中断。

4.3 两个以上组件同时升级

两个以上组件不会同时升级，因为升级是受控的，我们可以先升级一个组件，待其完全恢复后，再升级另一个组件。所谓完全恢复，是指控制平面和数据平面完全可用，且路由器所有组件处于一致的状态。

关于组件升级后的接口问题，各组件之间采用的跨进程通信方式是基于 WCF（Windows Communication Foundations）的，WCF 可以直接传递包括枚举值、字符串、列表、字典在内的高级数据结构，因此字典中增加新的字段不会影响原有字段的解析。

例如，MiniDB 的表中每个条目有生效、未生效、部分生效三种可能的状态。假定现在需要第四种薛定谔猫态“可能生效可能不生效”，就需要修改 SyncD 与 MiniDB 的接口。条目的状态用一个枚举值表示，现在增加了一个枚举值，原来的枚举值不受影响。首先升级 MiniDB，这时 SyncD 发来的所有状态仍然只可能是原来的三种；在 MiniDB 完全恢复后，再升级 SyncD，SyncD 重新启动后向 MiniDB 发送的状态就可能是四种中的任何一种了。

组件升级是有先后顺序的，必须先升级接口的调用者或内容的生产者，后升级接口的实现者或内容的消费者。在上面的假想例子中，如果先升级 SyncD，那么当新的 SyncD 启动后，就可能向 MiniDB 发送“薛定谔猫态”，而这是升级前的 MiniDB 所不认识的，可能带来无法预料的后果。

4.4 两个以上组件同时出错

两个以上组件同时出错从概率上是几乎不可能的，但为了严谨性仍然需要讨论：

- 如果 MiniDB 与 SyncD 几乎同时出现故障，当它们重启后将无法恢复重启前状态，此时管理平面看到的配置与路由器芯片中的实际配置是不同步的，而路由协议仍会以增量的方式推送新配置，由于 SyncD 不知道原来的配置，可能造成一些配置冲突。不过实际使用的 BGP、OSPF、源

MAC 学习等路由协议每隔几十秒会全量更新完整的配置，因此在全量更新之后，MiniDB 和 SyncD 中的状态就会成为完整的。

- 如果 MiniDB 与芯片 SDK 几乎同时出现故障，MiniDB 与芯片 SDK 都能从 SyncD 中恢复自身状态，控制平面中断时间由恢复较慢的芯片 SDK 决定，数据平面不会中断。
- 如果 SyncD 与芯片 SDK 几乎同时出现故障，SyncD 会首先从 MiniDB 中恢复状态，然后芯片 SDK 从 SyncD 中恢复状态（在 SyncD 恢复状态之前不会接受芯片 SDK 的访问，因此芯片 SDK 不会读取到不一致的状态），控制平面中断时间是 SyncD 与芯片 SDK 单独重启的恢复时间之和，数据平面不会中断。
- 如果 MiniDB、SyncD 与芯片 SDK 三者几乎同时出现故障，则 MiniDB 重启后配置为空，SyncD 重启后从 MiniDB 恢复到空配置，芯片 SDK 重启后从 SyncD 恢复到空配置，芯片 SDK 随即会用空配置覆盖路由器芯片中的配置，造成网络中断。不过，路由协议全量更新之后，数据平面会恢复正常，MiniDB、SyncD 和芯片 SDK 的内部状态也恢复到一致状态。因此这种极端情况下数据平面的中断时间取决于路由协议全量更新的频率。

第五章 系统评估

理论已经保证了容错软件架构的正确性，评估部分主要针对系统的性能，一方面是升级或出错时对控制平面和数据平面的影响时间，另一方面是整套系统对控制平面性能的影响。

5.1 组件重启的影响时间

测试项目包括控制平面和数据平面的影响时间。

1. 为模拟实际机架顶上的交换机（ToR switch）中表项的数量，我们通过一个路由协议客户端，插入下列表项：
   a) 在 ICAP 模糊匹配表内插入 1000 条匹配随机生成的 TCP 五元组的防火墙规则；
   b) 在二层转发表内插入 100 个 MAC 地址和物理端口的映射；
   c) 随机生成 10 个三层下一跳；
   d) 随机生成 10 个三层网络接口；
   e) 在三层前缀路由表和三层直接路由表中各插入 10000 个条目，随机引用三层下一跳和三层网络接口。

   这些表项在后续的实验中不会被发包工具发出的任何数据包匹配，仅仅是为了测试恢复表项所用的时间。
2. 用发包工具不断发送目的 IP 分别是 10.0.0.1 和 10.0.0.2、以太网数据包长度为 1518 字节的 UDP 包到 ServerSwitch 的 1 号端口，每条 UDP 流的速度为 10 Gbps（使用 40 Gbps 的以太网卡）。
3. 路由协议客户端插入一条路由规则，把目的 IP 为 10.0.0.0/24 的包转发到 2 号端口。
4. 在 2 号端口所连接的以太网卡处测量接收到的流量，应该是 20 Gbps。如果组件故障导致数据平面无法正常转发，流量应该是 0。我们每 0.5 秒测量一次流量，达到 15 Gbps（即 0.5 秒内收到了至少 7.5 Gb 的以太网帧）就认为对数据平面没有影响。
5. 杀掉欲测试的组件进程（模拟组件出错的情况），并立即通过路由协议客户端插入一条路由规则：把目的 IP 为 10.0.0.2/32 的包转发到 3 号端口。
6. 在 3 号端口所连接的以太网卡处测量接收到的流量，在控制平面恢复之前，新插入的路由规则尚未生效，3 号端口的流量应该是 0。新插入的路由规则生效后，流量应该是 10 Gbps。我们每 0.5 秒测量一次流量，达到

5 Gbps（即 0.5 秒内收到了至少 2.5 Gb 的以太网帧）就认为控制平面恢复了。

7. 路由协议客户端每 30 秒会做一次全量同步，即重新插入 10.0.0.0/24 和 10.0.0.2/32 这两条路由规则。

分别测试 MiniDB、SyncD、芯片 SDK 单独或共同“出错”（进程被杀掉）造成的数据平面和控制平面恢复时间，每种情况测试 10 次。

出错的组件	控制平面			数据平面		
	平均	最小	最大	平均	最小	最大
MiniDB	1.2	1.0	2.0	**0.0**	**0.0**	**0.0**
SyncD	7.0	6.0	8.5	**0.0**	**0.0**	**0.0**
芯片 SDK	64.3	61.5	73.0	**0.0**	**0.0**	**0.0**
SyncD + 芯片 SDK	72.1	68.5	78.5	**0.0**	**0.0**	**0.0**
MiniDB + 芯片 SDK	64.8	61.0	70.5	**0.0**	**0.0**	**0.0**
MiniDB + SyncD	1.5	1.0	2.5	30.6	30.0	31.5
MiniDB + SyncD + 芯片 SDK	65.2	62.0	75.5	65.2	62.0	75.5

表 5 组件重启时数据平面和控制平面的恢复时间（单位：秒）

表 5 的测试结果与 4.4 节的分析基本相符。主要结论是当 MiniDB、SyncD 与芯片 SDK 三个组件之任一因出错或升级而重启时，数据平面都不会中断，而控制平面中断的时间是可预测的。即使在 MiniDB 和 SyncD 同时出错的最坏情况下，控制平面和数据平面的中断时间仍然是可预测的。

5.2 对控制平面性能的影响

容错性是本软件架构的主要目标，但不能为了容错性而过分牺牲性能，即需要保证一定的表项插入速度。

采用对比测试的方法，测试组为我们的容错软件架构，由路由协议客户端不断插入条目；对照组为直接远程调用芯片 SDK，不断插入同样顺序、同样内容的

条目。我们对四种类型的表分别进行测试，初始均为空表。

- 精确匹配表：三层直接路由表
- 直接索引表：三层网络接口表
- 前缀匹配表：三层前缀路由表
- 模糊匹配表：ICAP，按优先级从高到低插入
- 模糊匹配表：ICAP，按优先级从低到高插入

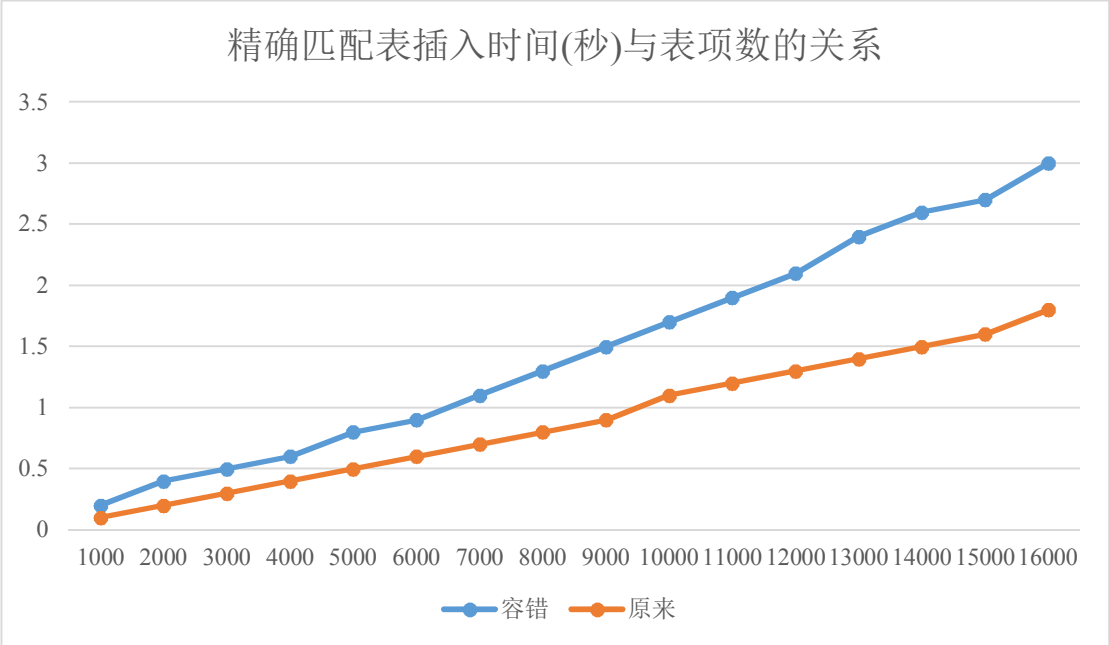

图 7 精确匹配表插入时间（秒）与表项数的关系

芯片 SDK 中精确匹配表插入时间是 O(n)的（n 为表项个数），而容错架构中由于需要维护堆，时间复杂度是 O(n log n)的，再加上进程间通信、检查主键等开销，使用容错软件架构后大约比直接使用芯片 SDK 慢一倍。

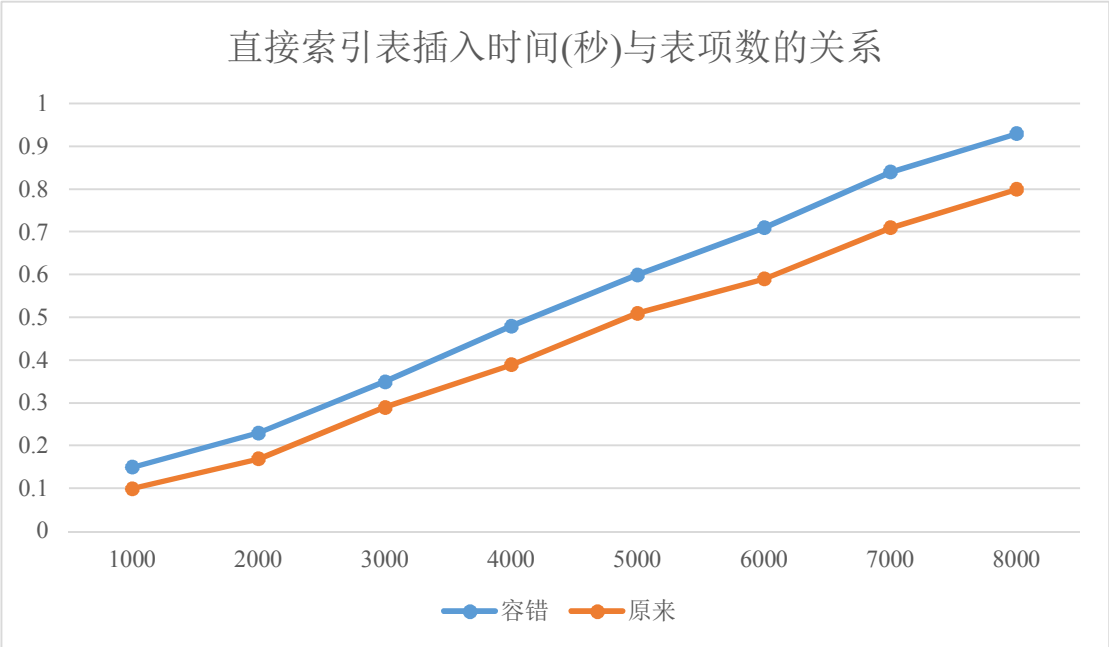

图 8 直接索引表插入时间（秒）与表项数的关系

芯片 SDK 中直接索引表的插入时间是 O(n)的，容错架构中也是 O(n)的，由于容错架构的各个组件运行在不同的进程中，构成了流水线，因此在 SDK 向硬件插入表项的同时，MiniDB 和 SyncD 可以继续处理新的表项，因此总的插入时间随表项数的增加没有明显的增长。增加的插入时间主要是“流水线”本身的延迟。

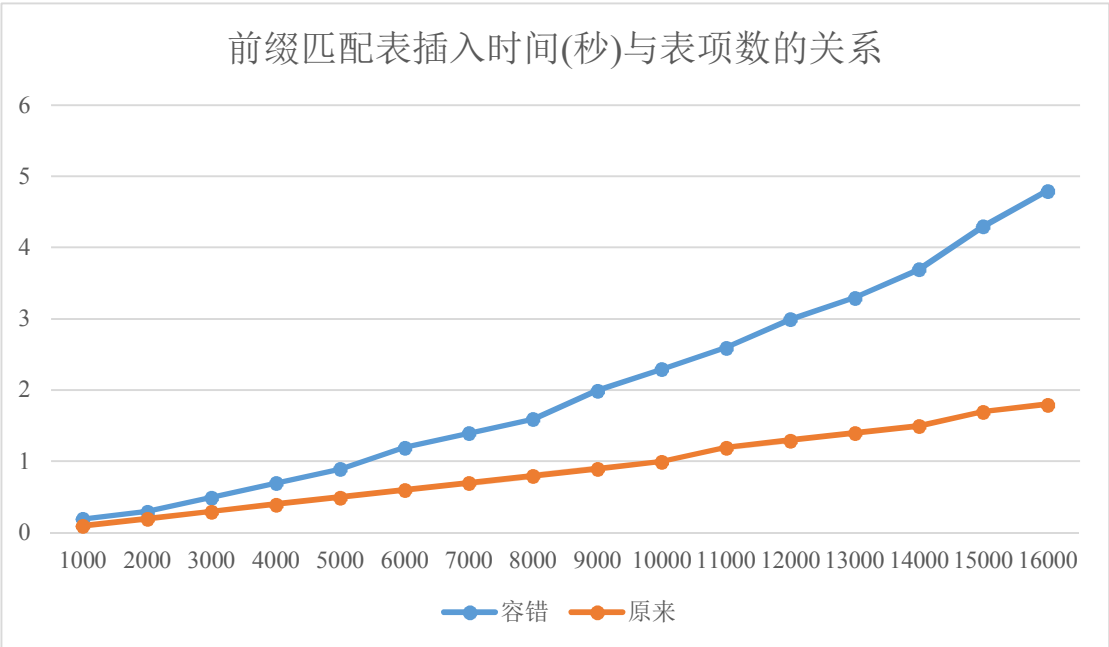

图 9 前缀匹配表插入时间（秒）与表项数的关系

芯片 SDK 中前缀匹配表的插入时间是 O(n)的，平均每秒能插入 1.1 万条前缀匹配表项。而容错架构中不仅需要维护堆，还要维护表项之间的冲突关系树，每插入一项就要遍历当前节点到树根的路径来检查冲突，O(n log n)时间复杂度中的常数较大，因此插入速度明显慢于芯片 SDK。

但在 1.6 万条前缀路由表项的情况下，仍然能在 5 秒内插入完毕，说明平均插入速度大于 3000 条/秒。数据中心内机架顶部交换机（ToR switch）的路由表项一般不会超过 10 K 的量级，而路由协议最快 30 秒更新一次，3000 条/秒的插入速度足够了。

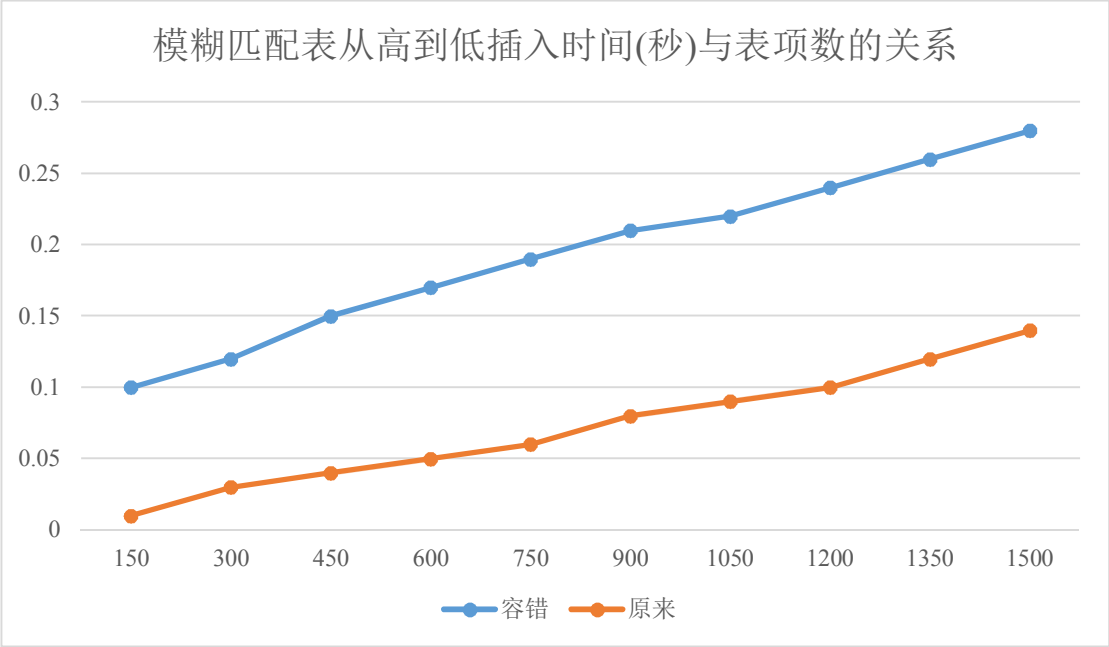

图 10 模糊匹配表从高到低插入时间（秒）与表项数的关系

模糊匹配表使用的是 TCAM，由于表项数有限，只能测试较少条目的情况。当按照优先级顺序插入时，由于每条硬件表项都是在末尾追加的，不需要移动原有的硬件表项，速度是足够快的。

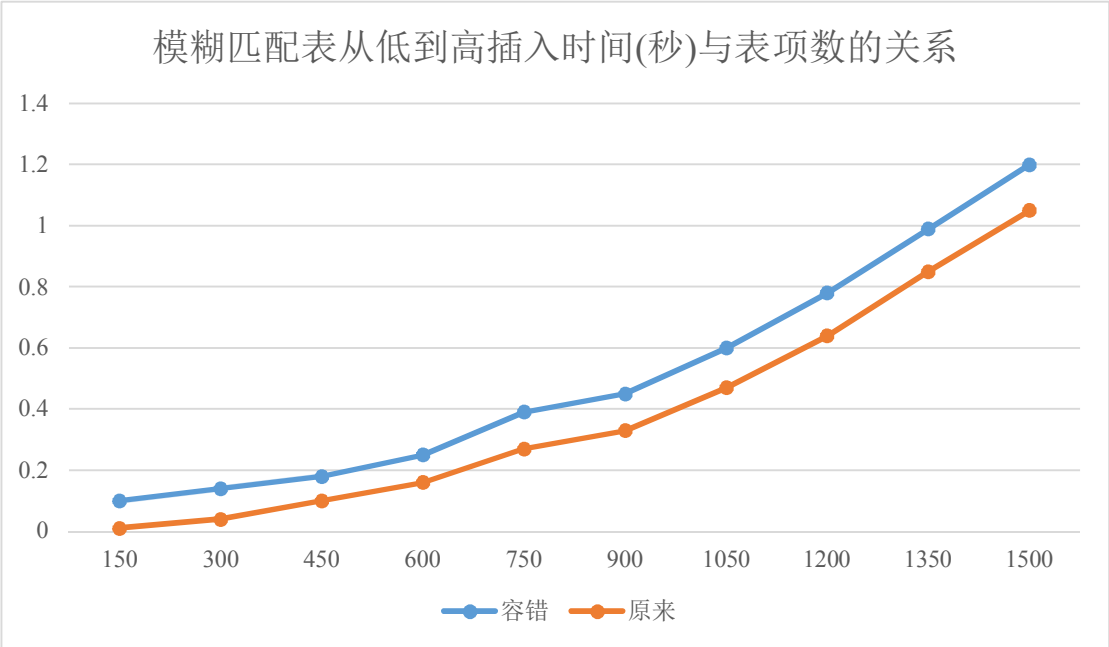

图 11 模糊匹配表从低到高插入时间（秒）与表项数的关系

模糊匹配表按照优先级从低到高的反向顺序插入时，每插入一个表项都需要把已有的每条硬件表项向后移动一个单元，再将新表项插入硬件表的第一个位置，故插入 n 条表项的时间复杂度是 $O(n^2)$。而且 DMA 访问硬件内存远远慢于访问本地内存和进程间通信，此时 DMA 是系统的瓶颈。图 11 的测试结果说明了在模糊匹配表插入的最坏情况下，容错软件架构没有成为系统的瓶颈。

综上，在机架顶上的交换机（ToR Switch）的应用场景下，本容错软件架构的性能是可以接受的。

第六章 总结

本文从两个方面提高了路由器软件架构的容错性：一方面，为每个路由协议客户端提供隔离的硬件资源抽象，使每个路由协议都认为自己在独占访问硬件资源，由同步服务按照优先级自动合并不同路由协议所写入的配置；另一方面，把路由信息库拆分为简单而稳定的轻量级路由信息库（MiniDB）和相对复杂的同步服务（SyncD），两个组件互为备份。

本路由器架构由管理平面和路由协议客户端、轻量级路由信息库、同步服务、芯片 SDK 等组件构成。在任一组件出错或升级时不会中断数据平面，而且能够在该组件重启后可预测的时间内自动恢复控制平面。

参考文献

[1] Clos, Charles (Mar 1953). “A Study of non-blocking switching networks”, Bell System Technical Journal 32(2): pp. 406 – 424.

[2] Guohan Lu, et al., “ServerSwitch: A Programmable and High Performance Platform for Data Center Networks”, In Proceedings of NSDI’11.

[3] Broadcom Trident II Switching Chip: http://www.broadcom.com/products/Switching/Carrier-and-Service-Provider/BCM56850-Series

[4] OpenFlow switch specification, version 1.3.0, 2013.

[5] Mark Reitblatt, et al., “FatTire: Declarative Fault Tolerance for Software-Defined Network”, HotSDN'13.

[6] Maciej Kuzniar, et al. “Automatic Failure Recovery for Software-Defined Networks (AFRO)”, HotSDN'13.

[7] An H. Lam, et al. “Fault-tolerant switch architecture”, US Patent 6925578 B2, Aug 2, 2005.

[8] Laprie, J.-C., et al. “Definition and analysis of hardware and software fault tolerant architectures”, Computer (Volume 23, Issue 7), pp. 39 – 51, July 1990.

[9] Cisco IOS Warm Reload & Warm Upgrade, Internet Technologies Division, September 2004: http://www.cisco.com/c/dam/en/us/products/collateral/ios-nx-os-software/high-availability/prod_presentation0900aecd8017a1a5.pdf

致谢

感谢指导我毕业论文的谭焜和华蓓导师，你们深厚的学术功底，渊博的学术造诣，严谨的工作态度，敏锐的科学洞察力，高昂的工作热情，坚韧的科研精神都让我深深折服。

感谢微软亚洲研究院无线与网络组的各位导师，是你们让我接触到了网络前沿的研究和先进的网络实验平台。特别要感谢谭焜导师不仅无私地教给我网络研究的方向和方法，还在我精神松懈的时候不断给我加油鼓劲。还要感谢张永光导师，是您吸引我走上网络研究的道路，并给我继续攻读博士的机会。

感谢科大 Linux 用户协会、少年班学院学生会技术部和格物致知社，在这里我遇到了一群志同道合的小伙伴和无数难得的实践机会。从第一条 Linux 命令开始，你们宽容地允许我一次又一次犯错，却一次又一次对我委以重任，在一个又一个团队里，从前端到后端，解决五花八门的问题，发现千姿百态的世界。

感谢郭家华、张成、周淼淼、肖世康等好朋友，是你们引领我走上计算机技术的道路。你们用知识和经验纠正我自以为是的认知，你们用对技术的热情照亮我前行的路。感谢贺羽、张伟等一同创业的小伙伴，你们让我不再把创业看作天边遥不可及的星矢，而是实实在在的一次探险、一种生活。

感谢科大的黄松筠班主任，在我从数学转向计算机和申请微软亚洲研究院两个关键节点上为我指引方向；感谢石家庄二中的雷勇、刘世洪班主任，你们不仅教给我数理知识，还教育我如何做人；感谢肖老师和石家庄桥西区奥数班，是你们让我获得了数学的启蒙，品尝到了人生第一次成功的喜悦。

最后，感谢含辛茹苦养育我的奶奶爷爷和妈妈爸爸，是你们帮我把屎把尿，教我识字读书，陪我挑灯夜战。感谢不免单薄，亲情值得用一生去感恩。

李博杰

2014 年 6 月 8 日