Милюткин М.Г.

СОЗДАНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ ДЛЯ ОПИСАНИЯ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ

(ФГОБУ ВПО «Поволжский государственный университет телекоммуникаций и информатики»)

В настоящее время маршрутизаторы производятся с аппаратно-фиксированным количеством протоколов маршрутизации. Сетевые инженеры делают все возможное при решении проблем маршрутизации и связности узлов сети, используя при этом не такой уж большой набор методов. Для IP-unicast маршрутизации это означает, что решения подобных проблем должны использовать только статическую маршрутизацию вместе со стандартизированными динамическими протоколами маршрутизации (RIP, OSPF, IS-IS, BGP) или коммерческим протоколом Cisco Systems EIGRP. Хотя BGP изначально был разработан как междоменный протокол маршрутизации, в настоящее время он используется в больших корпорациях в качестве внутридоменного протокола маршрутизации (не путать с использованием внутреннего BGP – IBGP, являющимся основным компонентом междоменной маршрутизации).

BGP полезен в качестве IGP потому, что он предоставляет более иерархичную структуру и дает возможность реализовать четко обозначенные административные границы — что часто необходимо в географически разбросанных и администратривно разнородных сетях. Не менее важен факт того, что традиционные IGP все основаны на алгоритмах кратчайших путей со строгими ограничениями в терминах контроля политик над маршрутизацией. Например, используя маршутизацию по методу кратчайшего пути, трудно реализовывать разные политики для разных направлений. Говоря кратко, BGP используется в качестве IGP не потому, что он настолько идеален, а потому, что он доступен, имеет ярко выраженные механизмы контроля политик, может быть использован для реализации административных границ, и, что не менее важно, он хорошо масштабируется. Не говоря уже о трудностях разработки и развертывания новых протоколов маршрутизации [1], или даже небольших изменений существующих протоколов. Если оставить в стороне трудоемкий процесс стандартизации, останется факт того, что очень трудно разрабатывать хорошо работающие протоколы с изощренным контролем политик, так необходимые в задачах внутридоменной маршрутизации во многих больших корпоративных сетях.

Однако, этот подход является не совсем корректным во многих отношениях, так как BGP не имеет гарантий сходимости. Более того, когда BGP используется в качестве IGP, политики маршрутизации имеют меньше ограничений по сравнению с внутридоменной маршрутизацией, где

стандартные политика типа «отдавать приоритет сначала маршрутам заказчика, затем маршрутам узлов и только затем маршрутам провайдеров» обеспечивают как минимум частичную защиту от расхождения протоколов [2]. Но если оставить в стороне расхождения протоколов, видно, что взаимодействие политик в BGP может привести к множеству стабильных решений, некоторые из которых были запланированы авторами политик, некоторые нет, и, когда система маршрутизации заклинивает в нежеланном маршруте, очень трудно производить ее отладку.

Как сделать из аппаратно фиксированного протокола гибкий протокол? Очевидно, что надо получить возможность программно его менять. Для этого можно предложить несколько подходов. Суть первого подхода состоит в том, чтобы выделить элементарные операции маршрутизатора и построить поверх него фреймворк на одном из высокоуровневых языков – для такой системной задачи подходит язык C. В дальнейшем, для программирования на каком-либо языке непосредственно логики протокола можно использовать этот фреймворк как библиотеку. В итоге имеем полную гибкость настройки протоколов маршрутизации. Минус этого подхода в том, что здесь необходимо иметь знания языков программирования C/C++ - то есть этот подход пригоден скорее для программиста, чем для сетевого инженера.

Второй подход стоит несколько ближе к не-программисту. Опять же берется набор атомарных действий машинного языка маршрутизатора и оборачивается в некий фреймворк с той разницей, что здесь фреймворк – не просто библиотека атомарных действий, обернутая в высокоуровневый язык, а нечто более содержательное, с примитивной логикой и пр. Для чего делать фреймворк сложнее? Для того, чтобы затем добавить сверху слой скриптования. Скриптовые языки создаются преимущественно для не-программирующей аудитории или для людей, которые понимают основы программирования и не хотят углубляться в детали. На скриптовом языке задаются некие параметры, исходные данные, которые затем обрабатываются фреймворком промежуточного уровня, создающим непосредственно логику протокола.

Третий подход является наиболее сложным и при этом наиболее гибким. Зададимся вопросом – а если сделать свой язык программирования протоколов?

Создание языка программирования, вообще говоря, представляет собой нетривиальную задачу, поскольку, прежде всего, требует проектирования, реализации и оптимизации соответствующего транслятора. Задачи, которые приходится решать с помощью компьютерных программ, крайне многообразны по своим предметным областям и для их решения требуются порой совершенно разные методики, разные подходы, что нашло свое непосредственное отражение в конкретных языках программирования. Каждый язык программирования представляет собой по сути инструмент, основанный на одной или нескольких таких методиках. Такие методики (подходы) называют также парадигмы программирования. На самом деле разные парадигмы программирования закладывали основу в разные языки программирования, каждый из которых создавался для решения своего типа задач.

Парадигма программирования (по-другому ее еще принято называть подход к программированию) в самом общем смысле – это фундаментальный стиль программирования вычислительных систем. На сегодняшний день существует множество различных парадигм [3], однако среди них можно выделить фундаментальные парадигмы, являющиеся базовыми для всех остальных, основанные каждая на своей математической теории (см. Табл. 1).

Таблица 1. Фундаментальные парадигмы программирования

Парадигма	**Теоретическое основание**
Объектно-ориентированное программирование	Машина Тьюринга
Императивное программирование	
Функциональное программирование	Лямбда-исчисления
Логическое программирование	Исчисление предикатов (логика первого порядка)

Фундаментальные парадигмы можно в самом общем смысле разделить на два различных по смыслу подхода – императивный и декларативный.

В декларативном подходе программист указывает что нужно сделать, не указывая при этом как это следует сделать. Таким образом, декларативный подход представляет собой уровень, независимый от особенностей платформы реализации. По этой причине декларативный подход наиболее предпочтителен для задач маршрутизации, так как освобождает сетевого инженера от необходимости прорабатывать детали выполнения нужной операции.

Таким образом, возможно определить протокол маршрутизации в декларативном высокоуровневом стиле. В данном случае необходимо лишь реализовать на маршрутизаторе транслятор для декларативного языка маршрутизации, чтобы запускать в действие таким образом заданный протокол.

Это позволит сетевому инженеру определять новый протокол маршрутизации самостоятельно и затем использовать его. Транслятор будут преобразовывать код такого языка напрямую в машинный код микропроцессора, под управлением которого работает маршрутизатор, либо в нечто более высокоуровневое: некий промежуточный язык, либо фреймворк. Такой подход даст максимальную гибкость в управлении маршрутизатором, однако он является наиболее затратным с точки зрения практического воплощения.

Во всех трех подходах имеется одна общая идея – все в итоге транслируется в машинный код микропроцессора маршрутизатора. Следовательно, важно тщательно исследовать элементарные операции и архитектуру такого процессора, так как именно над этими действиями будет строиться логика любого из вышеперечисленных подходов.

Литература

1. Полукаров Д.Ю. Нечеткая аппроксимация метрики протокола IGRP // Инфокоммуникационные технологии, 2006, Т. 4. – № 4. – с. 51 – 54

2. Lixin Gao, Jennifer Rexford, Stable Internet routing without global coordination. IEEE/ACM Transactions on Networking, 2001. – p. 681 – 692

3. http://en.wikipedia.org/wiki/Programming_paradigm