第一章　cap 章节

1.1　引言

SDN 的一个主要研究方向是可编程、高效的数据通路 (例如, OpenFlow1.3[1], OF-DPA[2], P4[3])。只有实现可编程性，SDN 数据通路才能支持各种不断发展的应用场景。同时，数据通路必须高效，能够满足高吞吐量和高成本效益等要求。在过去的几年中，多流表流水线已经成为 SDN 数据通路的一个关键部分 (例如, Domino[4], Forwarding Metamorphosis[5])。

然而，高效数据通路的一个问题是，它们必须经常以低级语言（甚至配置）进行编程，导致开发低效。例如，TCAM 对于实现高吞吐量至关重要，然而它不支持逻辑非。因此，SDN 的第二个主要研究方向是数据通路无关的、高级编程，并通过抽象来隐藏底层数据通路细节。在过去几年中，也出现了多个高级 SDN 编程模型（例如, Frenetic[6], Maple[7]）。

随着这两个研究方向的发展，一个基本问题则是：给定的高级程序能否在给定的低级数据通路上实现。对这一问题的良好理解将有助于高级 SDN 程序的设计和数据通路的设计。给定一个固定的数据通路（例如，一个固定的流水线结构，OF-DPA），数据通路的供应商可以提供相关的高级程序的指南。反过来，给定一组高级程序，可以来设计支持这些程序的最紧凑的数据通路。即使对于可重新配置的数据通路（例如, P4），由于重新配置是昂贵且耗时的，因此可以基于对该问题的理解指导设计更健壮的数据通路。考虑到所有可以在给定可编程数据通路上实现的高级程序可以看作是数据通路的容量，我们将这一基本问题定义为 SDN 数据通路编程容量问题。

然而，解决数据通路容量问题并非易事。考虑一个简单的数据通路，Simple-DP。如图 1.1所示，它是最简单的数据通路之一，由构成流水线的三个表组成，其中第一个表 (`t1`) 在源 IP 上匹配，并且可以跳到以下两个表，这两个表都在目标 IP 上匹配。

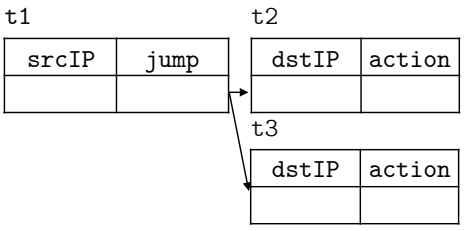


图 1.1: 一个简单的数据通路示例: Simple-DP

我们再考虑下面两个简单的高级 SDN 程序，它们都采用算法性、事件驱动的编程模型

（下面会给出有关编程模型的更多详细信息）。我们可以发现第一个程序可以通过 Simple-DP 实现，而第二个程序则不能。

```
//  Routing Function: secureL3Route
L1: def secureL3Route(Addr srcIP, Addr dstIP):
L2:   if srcIP == 10.0.0.1:
L3:     return Forward(port=shortestPath(dstIP))
L4:   else:
L5:     return Drop();
```

```
//  Program: twoHostL3Route
L1: def twoHostL3Route(Addr srcIP, Addr dstIP):
L2:   if srcIP == 10.0.0.1:
L3:     return Forward(port=shortestPath(dstIP))
L4:   elif srcIP == 10.0.0.2:
L5:     return Forward(port=securePath(dstIP))
L6:   else:
L7:     return Drop();
```

尽管前面的数据通路和高级程序是最简单的，但它们已经可以说明解决 SDN 数据通路容量问题并不明显。由于需要实现多个服务，一般数据通路和高级程序可能要复杂得多，因此它们在分析中可能会带来严峻的挑战。本章的目标是试图得到一个系统的方法来解决 SDN 数据通路容量问题。

本文的贡献可以概括如下。首先，我们提出了一个统一的特征函数空间来消除程序结构和流水线布局的复杂性和不一致性。其次，我们在这个函数空间中定义了一个比较运算，用来判断一个高级程序是否可以在给定的流水线上实现。

1.2 相关工作

高级 SDN 程序编译器：过去几年，人们提出多种允许程序员用高级语言编写 SDN 程序，并将这些程序编译成流表的系统。因为它们需要检查高级程序到交换机流表的转换，因此这些系统与本章工作相关。我们将这些系统分为两类：无层和分层。

无层系统 (如 SNAP[8], FML[9], FlowLog[10], Maple[7])，要求程序员通过编写数据包处理函数的形式，指定数据包的转发行为。然后 SDN 控制器使用这些函数来配置和更新网络状态。这样的系统启发了流水线容量定理中函数的概念。但是，这些系统在没有实际执行的情况下无法验证提交的函数是否可以写入给定的流水线结构。

分层系统如 Frenetic 家族 (Frenetic[6], Pyretic[11]) 提供了一个两层编程模型，其中控制器程序指定感兴趣的事件，然后在这些事件发生时通过计算新的网络策略来响应。同样，这样的系统无法验证给定控制器程序的输出是否可以写入交换机的流水线。

流水线定制语言：流水线定制语言 (如 P4[3], PISCES[12], Concurrent NetCore[13]) 和我们的流水线容量定理之间有一些相似之处，同样对流水线行为进行了分析以及保证。例如，Concurrent NetCore 的类型系统确保用于部署到流水线的任何程序都具有某些属性[13]，而 PISCES 的交换机规范允许编译器分析流水线并优化其性能[12]。然而保证流水线属性或提高性能与验证编译是否可行是不同的。

流水线设计：流水线设计方案，如 Jose 等人[14] 考虑的将数据包程序编译到可重构交换机、Sun 等人[15] 的软件定义流表流水线、FlowAdapter[16] 和 Domino[4] 显然与我们的流水线容量定理相关，因为它们检查硬件约束下的流水线布局设计。然而，Jose 等人，Sun 等人，和 FlowAdapter 着重于将逻辑查找表或多流表流水线映射到物理表，而我们的流水线容量定理侧重于通用程序，而 Domino 考虑的是较弱的硬件约束条件 (如有状态操作)。

1.3 模型

我们首先给出高级 SDN 程序和低级数据通路的模型。由于 SDN 的主要关注点是路由，因此我们这里将高级 SDN 程序称为路由函数。由于多流表流水线是 SDN 数据通路的最先进技术，因此我们这里将流水线作为数据通路对象进行研究。

1.3.1 路由函数模型

路由函数：我们将一个路由函数表示为 f。我们考虑它是一个逻辑上集中的、确定的函数，并用高级语言编写。SDN 控制器对进入该控制器网络的每个数据包逻辑上执行该函数[7]，以计算出该数据包的全网范围的路由。

每个 f 在数据包上的执行要读取数据包的一组属性（称为匹配字段）$\mathcal{M} = \langle m_1, ..., m_n \rangle$ (例如, `<srcIP, dstIP, ...>`)。我们用 M 表示数据包匹配字段在 $\mathcal{M}$ 中的一个子集。此外，我们用 $dom(M)$ 来标识一组匹配字段 M 的域。f 的执行会从一组（对数据包的）有效操作 $\mathcal{R}$ 返回一个路由操作，(例如, `Drop, Forward(port=2)`):

$$f : dom(\mathcal{M}) \to \mathcal{R}.$$

这样的函数所在的空间标识为 $\mathcal{F}$。

示例：我们用如下的路由函数 `onPkt` 来演示我们路由函数模型的一些关键特性。

```
\\  Routing function: onPkt
    Map hostTbl[key: dstIP, value: switch]
    Map condTbl[key: (dstIP, port), value: cond]
    Map routeTbl[key: (switch, cond), value: outPort]
L1: def onPkt(Type ethType, Addr srcIP, Port srcPort, \
          Addr dstIP, Port dstPort):
L2: if (ethType != IPv4):
```

```
L3:   return Drop()
L4: if (verify(srcPort, srcIP)):
L5:   dstCond = condTbl[dstIP, dstPort]
L6:   dstSw = hostTbl[dstIP]
L7:   return Forward(port = routeTbl[dstCond, dstSw])
L8: return Drop()
```

具体来说, `onPkt` 读取匹配字段 $\mathcal{M} = \langle \texttt{ethType}, \texttt{srcIP}, \texttt{srcPort}, \texttt{dstIP}, \texttt{dstPort} \rangle$ 并将 $\mathcal{M}$ 域中的每个值映射到 $\mathcal{R} = \{\texttt{Drop()}, \texttt{Forward(port=x)}\}$ 中的路由操作。虽然我们将 `onPkt` 以命令式函数写出，需要强调的是我们的模型是完全通用的，并且没有指定编程范式。

具体地说，`onPkt` 的前三行声明了键值表：`hostTable` 和 `condTable`。它们分别把每个 IP 地址和一个连接交换机以及主机条件 (例如, 身份验证状态) 进行关联，而 `routeTable` 将（交换机、条件）映射到其转发端口。在 `onPkt` 的主体部分上，`L2` 和 `L3` 检测并丢弃非 IPv4 流量，而 `L8` 则丢弃来自未验证的终端的流量。如果验证通过，`L5` 到 `L7` 设置 `dstCond` 和 `dstSw` 变量，然后基于这两个变量从 `routeTbl` 返回路由操作。

路由函数数据流图：由于一般的路由函数可以具有任意的、复杂的控制结构，我们将路由函数转换为数据流图（Dataflow Graph，DFG）以更好地表示其结构。我们把 f 的 DFG 表示为 G_f。

具体来说，要计算 f 的 G_f，我们必须删除 f 中所有的控制流依赖。这些依赖通过以下转换被删除：

- 通过将 f 转换为静态单赋值形式（Static Single-Assignment，SSA）来移除赋值语句顺序依赖；
- 通过将分支的条件值赋给 guard 变量来移除分支，并将对这些 guard 的依赖性附加到 `if` 和 `else` 块中的所有语句；
- 通过将程序循环转换为黑盒函数来删除它们，黑盒函数的输入为循环中读取的所有变量，输出为循环中写入的所有变量。

例如，我们的示例路由函数 `onPkt` 转换如下：

```
L1: def onPkt(...):
L2: g0 = (ethType != IPv4)
L3: if  g0: return Drop()
L4: g1 = verify(srcPort, srcIP)):
L5: if  g1: dstCond = condTbl[dstIP, dstPort]
L6: if  g1: dstSw = hostTbl[dstIP]
L7: if  g1: return Forward(port = routeTbl[...])
L8: if !g1: return Drop()
```

注意 `onPkt` 的 `L2` 处的 `if` 语句已经从条件表达式变成了 guard 变量 `g0`，`g0` 又被附加给了 `L3`，这一行之前是在 `if` 块内的。

在给出了这个转换后，我们定义 f 的 G_f:

定义 1 *一个路由函数 f 的数据流图 DFG $G_f = (V_f, E_f)$ 是一个从转换的 f 生成的点带权的有向无环图，其中:*

- *V_f 中的每个点 v_f 是 f 的一个变量;*
- *一个 v_f 的权重是它的域值范围大小;*
- *当两个变量之间存在一条 E_f 中的有向边时，意味着源变量出现在目的变量的赋值声明中。*

作为示例，我们给出 `onPkt` 的 DFG 如下:

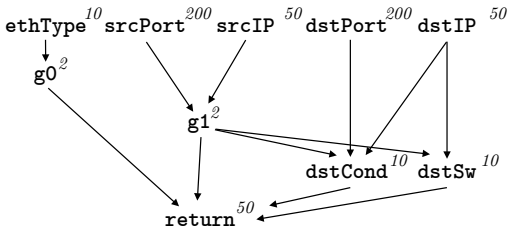


图 1.2: 路由函数 `onPkt` 的 DFG $G_{\texttt{onPkt}}$

可以看到，点 `dstSw` 是由其赋值声明中的两个变量 `g1` 和 `dstIP` 派生得到的。该点的权重 10 表示 `dstSw` 的域值范围大小。

1.3.2 流水线模型

我们考虑目前最新的数据通路设计：多流表流水线架构。我们首先对一个流水线 pl 中的一个表 t 进行建模，然后给出对于流水线的清晰定义。

流水线中的表：每个流水线中的表 $t \in pl$ 是精准匹配的 match-action 表。t 的匹配操作可以是：（1）一个路由操作；（2）对 t 的输出寄存器 $r(t)$ 的写操作以及对 pl 中的后续表的跳操作；（3）对 pl 中的后续表的简单跳操作。并不是所有的 t 都输出路由操作。我们把输出路由操作的 t 称为出口表。

每个 t 匹配着一组输入 $I(t)$。输入 $I(t)$ 包含数据包的匹配字段 $m_i \in \mathcal{M}$ 和前序表的输出寄存器 $r(t)$。t 的关键限制是它可以包含的最多规则数量和 $r(t)$ 的比特位长度，在这里我们对应地表示为 $maxrules(t)$ 和 $bits(r(t))$

流水线：一个流水线 pl 是关于表集合 $\{t_i\}$ 的单根有向无环图。pl 中的一条边 (t_i, t_j) 表示到达 t_i 的数据包可以跳转到 t_j。

每个通过 pl 的数据包从 pl 的根节点出发，然后在 pl 中被处理，最终传到出口表。因此，每个包在 pl 中传输的过程可以映射到 pl 中的一条路径，以及对该数据包进行的路由操作（属于 $\mathcal{R}$）。

一个数据包在 pl 中通过的路径和它的出口表输出的操作由这个数据包的匹配字段组 $\mathcal{M}$ 在每个 $t_i \in pl$ 中的匹配情况决定。基于此，pl 也可以概括为一个从 $dom(\mathcal{M})$ 到 $\mathcal{R}$ 的映射，这个映射依赖于 pl 的内容，即流表。

我们把所有流水线 pl 所在的空间标识为 P。

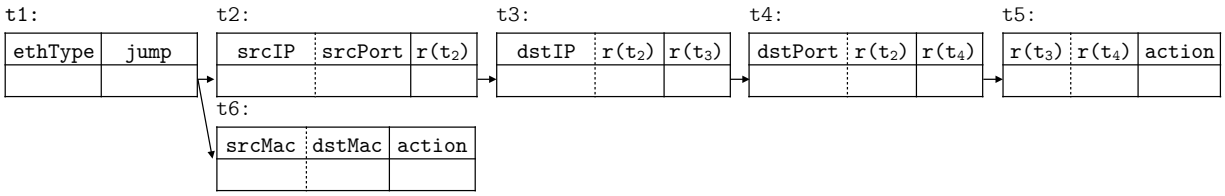


图 1.3: 示例数据通路, ExampleDP

示例：我们现在给一个示例流水线 ExampleDP（如图 1.3所示），来说明我们的流水线模型。要注意的是在这个例子中，一个流表的匹配列在它的左边，然后写入到它右边的寄存器中，而表中字段 output 表示这个表包含了输出路由操作。

考虑流表 t2 ∈ ExampleDP。t2 是个精准匹配表，它的输入 I(t2) 是 srcIP 和 srcPort，它的输出寄存器是 r(t2)。

关于 t2 的关键计算限制是它的最大规则数 $maxrules$(t2)) 和他的输出寄存器的大小 $bits$(r(t2))。

1.4 主要结果以及相关证明

基于函数模型和流水线模型，我们现在给出关于函数 f 是否可以由流水线 pl 实现的主要结果。然后我们再给出结果的相关证明。关键的一些符号在表1.1中写出以供参考。

1.4.1 概述

本节的目标是得到一种系统方法来验证路由函数 f 是否可以由流水线 pl（我们称之为 $f \Longrightarrow pl$）实现。存在的一个主要挑战是路由函数和流水线的表示方式是不同的，并且这两种表示方式都具有巨大的复杂性和多样性。可以想象将每个路由函数 f 看作函数空间 $\mathcal{F}$ 中的一个点，将每个流水线 pl 看作函数空间 $\mathcal{P}$ 中的一个点。

我们的主要贡献是引入了一个创新的、统一的、规范化的函数空间 $\mathcal{C}$，我们称为特征函数空间。每个路由函数 f 通过将 τ 映射到特征函数 $\tau(f) \in \mathcal{C}$ 来表示 f 的计算负载。另一方面，每个流水线 pl，则被映射到一组特征函数 $\kappa(pl) \subset \mathcal{C}$，表示这个流水线的一组计算能力。图 1.4给出了这种映射结构。

符号	定义
路由函数相关符号	
f	路由函数
$\mathcal{F}$	路由函数空间
m_i	数据包的匹配字段
$\mathcal{M}$	$\forall\ m_i$ 的集合
$dom(M)$	M 中有效值的域
$\mathcal{R}$	$\forall$ 有效路由行为的集合
流水线相关符号	
pl	流水线
$\mathcal{P}$	流水线函数空间
t_i	流水线中的表
$r(t_i)$	t_i 的输出寄存器
$bits(r(t_i))$	t_i 的输出寄存器 bit 位长度
$I(t_i)$	t_i 的输入
$maxrules(t_i)$	t_i 可以包含的最大 # 规则数

表 1.1: 在我们 “主要结果以及相关证明” 一节中使用的符号标识列表。

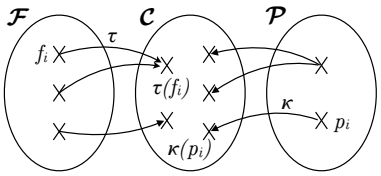


图 1.4: 空间 $\mathcal{F}$, $\mathcal{P}$ 和 $\mathcal{C}$ 及其它们之间的映射表示。

鉴于 $\tau(f)$ 和 $\kappa(pl)$ 都定义在了同一空间 $\mathcal{C}$（其中 $\tau(f)$ 对应点，$\kappa(pl)$ 对应一组点），人们就可以通过比较 $\kappa(pl)$ 中的每个元素与 $\tau(f)$，从而判断这个计算负载是否可以被一个计算能力 “覆盖”，进而导出了我们的基本容量理论，即 $\exists\ c \in \kappa(pl) \geq \tau(f), f \Longrightarrow pl$。

1.4.2 特征函数

我们首先来定义一个通用的特征函数 c。

定义 2 *一个特征函数 c 是一个从数据包的匹配字段的子集 M 到一个向量的映射：*

$$c(M) \triangleq < scope(M),\ ec(M) > .$$

我们将 $c(M)$ 的向量的两个部分对应地称为 $c(M)[scope]$ 和 $c(M)[ec]$。

给定两个特征函数，我们可以比较它们。

定义 3 我们定义 c_i 支配 c_j, 标识为 $c_i \geq c_j$ 如下:

$$c_i \geq c_j \triangleq \forall\, n \in \{scope, ec\},$$
$$\forall\, M \in 2^{\mathcal{M}},\ c_i(M)[n] \geq c_j(M)[n].$$

为了确保我们的容量理论，我们需要将一组特征方程和一个单独的特征方程作比较。

定义 4 一组特征方程 C_i 支配一个特征方程 c_j, 标示为 $C_i \trianglerighteq c_j$，如果存在一个 $c_i \in C_i$ 支配 c_j

$$C_i \trianglerighteq c_j \triangleq \exists\ c_i \in C_i : c_i \geq c_j.$$

1.4.3 路由函数的特征

给定特征函数的概念，我们现在导出路由 f 的特征函数，表示为 $\tau(f)$。

定义 5 关于数据包匹配字段 M 的一个路由函数的特征方程的范围（*scope*）定义为 M 中有效值域的大小:

$$\tau(f)(M)[scope] \triangleq dom(M)$$

$\tau(f)(M)[ec]$ 是我们根据 “f 等价” 概念建立的属性:

定义 6 我们定义 f 等价（标识为 $\sim_f$），为 M 中两个值 $v_i(M)$ 和 $v_j(M)$ 之间的一种联系，即不能被 f 所区分:

$$v_i(M) \sim_f v_j(M) \triangleq \forall\ v_k(\mathcal{M} - M) \in dom(\mathcal{M} - M),$$
$$f(v_i(M), v_k(\mathcal{M} - M)) = f(v_j(M), v_k(\mathcal{M} - M)).$$

我们关于 f 等价的定义可以自然地导出 f 等价类的定义。

定义 7 一个 f 等价类（标识为 $[v_i(M)]_f$），是一个给定的 M 中的变量 $v_i(M)$ 的所有和它 f 等价的变量的组:

$$[v_i(M)]_f \triangleq \{v_j(M) \in dom(M) : v_i(M) \sim_f v_j(M)\}.$$

计算等价类的数量可以得出 f 等价类数的概念。

定义 8 M 中 f 等价类的数量（标识为 $|dom(M)/\sim_f|$），是 M 的一组等价类的基数。

现在来看我们关于 $\tau(f)(M)[ec]$ 的定义。

定义 9 M 的路由函数特征函数的 ec（*equivalent class*）是 M 的 f 等价类集合的基数。

$$\tau(f)(M)[ec] \triangleq |dom(M)/\sim_f|$$

定义 10 *路由函数的特征函数 $\tau(f)$ 表示 f 的计算负载:*

$$\tau(f)(M) \triangleq (dom(M),\ |dom(M)/\sim_f|).$$

虽然 $\tau(f)$ 非常强大，但它是不切实际的，因为直接计算等价类数需要非常大的计算。因此，我们通过定义路由函数 $\tau_G(f)$ 的边界特征函数来限制 $\tau(f)$，该函数很容易从 f 的 DFG 导出。该函数刻画了 f 的计算负载的一个上界：$\tau_G(f)$ 支配 $\tau(f)$。

$\tau_G(f)[scope]$ 的定义如之前所述。然而，我们不计算 $\tau_G(f)[ec]$，而是用 G_f（即 f 的 DFG）中特定的顶点割值来确定其上限。我们现在开始说明这个割值。

定义 11 *设 $V_f(M)$ 是 G_f 中的点 $m_i \in M$，$D_f(M)$ 是从 $V_f(M)$ 派生的 G_f 中的点。M 的顶点最小割集，$G_f.vertexMinCut(M)$，是将 $V_f(M)$ 从 $D_f(\mathcal{M}-M)$ 中分离出来的最小割集中顶点的权重的乘积。*

给定这个割集，我们定义 $\tau_G(f)$ 如下:

定义 12 *一个路由函数的特征函数 $\tau_G(f)$ 刻画 f 的计算负载上限; $\tau_G(f)$ 支配 $\tau(f)$:*

$$\tau_G(f)(M) \triangleq (dom(M),\ G_f.vertexMinCut(M)).$$

示例：现在用我们的示例路由函数 `onPkt` 来说明这些概念。

考虑 `onPkt` 的匹配字段 `srcIP` 和 `srcPort`。每个都只被读取一次：在 `L4` 上，由布尔函数 `isVerified` 读取。因此，虽然 `srcIP` 和 `srcPort` 可能分别有许多 f 等价类，(`srcIP`, `srcPort`) 只有两个：`isVerified` 的值计算为 0 和它的值计算为 1。

假设 `onPkt` 是一个小型商业网络的路由函数，其外层是一个连接有 50 个主机的 NAT，每个主机运行着一些受限制的应用，这个限制是指只能使用 200 个标准端口。给定这个条件下，$dom(\texttt{srcIP}, \texttt{srcPort}) = 10000$，以及 $\tau(\texttt{onPkt})(\texttt{srcIP}, \texttt{srcPort}) = (10000, 2)$。

虽然 (`srcIP`, `srcPort`) 的等价类数是直接的，但 `onPkt` 输入的大多数其他子集的等价类数并不明显。因此，我们将 $\tau(\texttt{onPkt})$ 用 $\tau_G(\texttt{onPkt})$ 约束，它的计算使用 `onPkt` 的 DFG（$G_{\texttt{onPkt}}$）。如图 1.5所示。

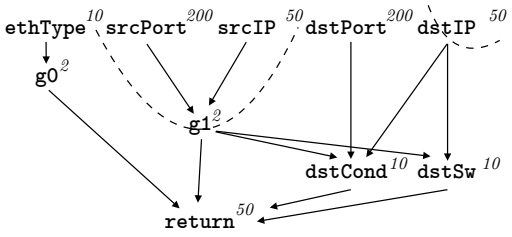


图 1.5: 路由函数 `onPkt` 的 DFG $G_{\texttt{onPkt}}$ 和割集 (`srcIP`, `srcPort`, `dstPort`)

我们考虑 `onPkt` 的输入为 (`srcIP`, `srcPort`, `dstIP`) 的等价类数量。我们取它们的点和 `onPkt` 的其他输入: (`ethType`, `dstPort`,`g0`, `dstCond`, `return`) 派生的每个点之间的 $G_{\texttt{onPkt}}$ 中的顶点最小割集。图 1.5中用虚线表示了这个顶点最小割集。

该割集中的顶点 (g1, dstIP) 的权重为 50 和 2，因此 $\tau_G(\texttt{srcIP},\texttt{srcPort},\texttt{dstIP}) = (50000,\ 100)$。

1.4.4 流水线的特性

我们现在定义流水线 pl 的特征函数集 $\kappa(pl)$。我们从定义通过流水线 pl 的路径 ρ 开始。

定义 13 *一个路径 ρ, 在 pl 中是一个通过 pl 的有向无环图的路径 $\langle t_1, ..., t_n\rangle$ ，其中 t_1 是 pl 的根表，t_n 是 pl 的出口表。*

作为示例，ExampleDP 有两个路径 $\langle \texttt{t1},\texttt{t2},\texttt{t3},\texttt{t4},\texttt{t5}\rangle$ 和 $\langle \texttt{t1},\texttt{t6}\rangle$，我们用 ρ_{L2} 和 ρ_{L3} 分别表示。

我们定义，$\forall\ \rho \in pl,\ \kappa_\rho(pl)$ 是一个流水线中一条路径的特征方程。

定义 14 *pl 的特征方程组 $\kappa(pl)$ 是 $\forall\ \rho \in pl$ 的集合:*

$$\kappa(pl)(M) \triangleq \{c \in \mathcal{C} : c = \kappa_\rho(\rho)\ \forall\ \rho \in pl\}.$$

我们现在通过引入以下定义来构造路径 ρ 的特征函数：

定义 15 *表 $t_i \in \rho$ 的输入闭包 $\bar{M}_\rho(t_i)$ 是 t_i 可以获得信息的输入集合：*

$$\bar{M}_\rho(t_i) \triangleq \{m_i \in \mathcal{M} : m_i \in I(t_i)\ \vee \\ m_i \in \bar{M}_\rho(t_j)\ s.t.\ r(t_j) \in I(t_i)\}.$$

定义 16 *ρ 的 M 的闭包 $\bar{\bar{M}}_\rho(M)$ 是由 $t_i \in \rho$ 与输入闭包 M 组成的。*

$$\bar{\bar{M}}_\rho(M) \triangleq \{t_i \in \rho : \bar{M}_\rho(t_i) = M\}.$$

通过这些定义，我们现在定义 ρ 的特征方程如下：

定义 17 *ρ 的特征函数 $\kappa_\rho(\rho)$ 表征了 ρ 的计算容量。*

$\kappa_\rho(\rho)[scope]$ 是 ρ 可以读取的 M 的最大数量，$\kappa_\rho(\rho)[ec]$ 是 ρ 可以分辨的 M 的等价类的最大数量。

$$\kappa_\rho(\rho)(M) \triangleq \\ \begin{cases} \bar{\bar{M}}_\rho(M) \neq \emptyset & (min[maxrules(t_i) : t_i \in \bar{\bar{M}}_\rho(M)], \\ & \quad min[2^{bits(r(t_i))} : t_i \in \bar{\bar{M}}_\rho(M)]) \\ \bar{\bar{M}}_\rho(M) = \emptyset \wedge \exists\ m_i \in M : & \\ m_i \notin \bigcup_{t_i \in \rho} \bar{M}(t_i, \rho) & (1,1) \\ otherwise, & (\mathrm{T},\mathrm{T}). \end{cases}$$

示例：与之前一样，我们使用示例流水线 ExampleDP 直观地说明流水线的特征函数。

ExampleDP 包含两个 ρ: ρ_{L2} 和 ρ_{L3}。考虑表 t4，仅包含在 ρ_{L3} 中。由于 t4 读取 dstIP 和 r(t2)，而 t2 依次读取 srcIP 和 srcPort，因此输入闭包 $\bar{M}_{\rho_{\text{L3}}}(\text{t4})$ 为 (srcIP, srcPort, dstIP)。闭包 $\bar{\bar{M}}_{\rho_{\text{L3}}}(\text{srcIP}, \text{srcPort}, \text{dstIP})$ 关于 t4 在 ρ_{L3} 的输入是 {t4}。

因此，$\kappa_{\rho_{\text{L3}}}(\bar{M}_{\rho_{\text{L3}}}) = \kappa_{\rho_{\text{L3}}}(\text{srcIP}, \text{srcPort}, \text{dstIP}) = (maxRules(\text{t4}), 2^{bits(r(\text{t4}))})$. 在 t4 有 2^{20} 规则数并且一个 16bit 输出寄存器这种情况下, $\kappa_{\rho_{\text{L3}}}(\bar{M}_{\rho_{\text{L3}}}) = (2^{20}, 2^{16})$.

进一步，我们考虑 ExampleDP 的匹配域的子集 (srcMac, dstMac)。ρ_{L3} 并不包含 srcMac 以及 dstMac，因此它只可以实现不包含 srcMac 以及 dstMac 的函数（或赋值为常数）。常数只有一个值，一个等价类。因此，对于任何输出包含 srcMac，$\kappa_{\rho_{\text{L3}}}$ 的值为 $(1, 1)$。

最后，我们考虑 ExampleDP 的匹配域的子集 (srcIP, srcPort)。srcIP 和 srcPort 都被 ρ_{L3} 所读，但 (srcIP, srcPort) 并不是任何 $t_i \in \rho_{\text{L3}}$ 的输入闭包。在这种情况下，在判断数据通路的实现问题上不需要考虑 (srcIP, srcPort)，因此，$\kappa_{\rho_{\text{L3}}}(\text{srcIP}, \text{srcPort}) = (\text{T}, \text{T})$，表示我们在和路由函数的 τ 比较时可以跳过。

1.4.5 数据通路编程容量理论

结合前面关于路由函数和流水线以及它们特征函数的定义，我们最终得到了我们的核心结论：给定的 f 是否可以在给定的 pl 中实现的一个充分条件。

定理 1 (流水线实现定理) *当 $\kappa(pl)$（即 pl 的特征函数集）支配 $\tau(f)$（即 f 的特征函数）时，一个路由函数 f 可以在流水线 pl 上实现。我们有:*

$$\kappa(pl) \trianglerighteq \tau(f) \Rightarrow f \Longrightarrow pl.$$

作为推论，因为 $\tau_G(f) > \tau(f)$，流水线实现定理可以推广到 $\tau_G(f)$。

示例：我们使用 onPkt 和 ExampleDP 来说明我们的流水线实现定理。具体来说，我们的流水线实现定理指出 $\kappa(\text{ExampleDP}) \trianglerighteq \tau(\text{onPkt}) \Rightarrow \text{ExampleDP} \Longrightarrow \text{onPkt}$。

进一步，当 $\kappa_\rho(\rho_{\text{L2}}) > \tau_G(\text{onPkt})$ 或者 $\kappa_\rho(\rho_{\text{L3}}) > \tau_G(\text{onPkt})$ 时，$\kappa(\text{ExampleDP}) \trianglerighteq \tau(\text{onPkt})$ 为真。我们通过比较每对特征函数给出的每个向量的每个分量来验证每个条件。举例来说，$\tau(\text{onPkt})(\text{srcIP}, \text{srcPort}, \text{dstIP}) = (50000,\ 100)$, $\kappa_\rho(\rho_{\text{L3}})(\text{srcIP}, \text{srcPort}, \text{dstIP}) = (2^{20}, 2^{16})$，因此输入集 (srcIP, srcPort, dstIP) 不会阻碍 onPkt 在 ρ_{L3} 上实现。

严密性：虽然定理只提供了一个充分条件，但在一些情况下可以建立更严密的结果，即充分必要的条件。具体来说，我们有以下结果：

定义 18 *当 pl 的 DAG 没有分支时，我们称 pl 为无分支流水线。*

定理 2 (无分支流水线实现定理) *如果 pl 是无分支流水线，且 pl 的表容量足够大，并且每个匹配字段 $m_i \in \mathcal{M}$ 正好出现在 pl 的每一张表中，$\kappa(pl) \trianglerighteq \tau_G(f) \Leftrightarrow f \Longrightarrow pl$.*

1.4.6 证明

流水线实现定理的证明

我们现在提出流水线实现定理的证明。这些证明的结构如下。首先，我们建立一种编码机制对 $m \in \mathcal{M}$ 进行编码，并给予的足够信息，以完全执行给定的 f。其次，在 $\kappa(pl) \trianglerighteq \tau_G(f)$ 前提下，证明使用我们的编码的流水线可以在 pl 中实现 f。最后，我们通过一个扩展：如果 $\kappa(pl) \trianglerighteq \tau(f), f \Longrightarrow pl$ 来证明 $\tau_G(f) > \tau(f)$。

我们将基于 f 的 G_f 的 $G_f.vertexMinCut(M)$ 中的点。

定义 19 *最小割集 $\mu_f(M)$ 是 f 的 G_f 用 $G_f.vertexMinCut(M)$ 分割得到的点。*

给定 $\mu_f(M)$ 的值为 $v_i(\mu_f(M))$ 且 $\mu_f(M)$ 的值的域为 $dom(\mu_f(M))$。

引理 1 *给定 $G_f.vertexMinCut(M)$，我们可以在不知道给定的值 $v_i(\mu_f(M))$ 的情况下计算 f。*

虽然 $\mu_f(M)$ 可以有效表示 M 的值，$v_i(M)$，但我们可以通过引入码字（codeword）的概念来对它进行压缩，从而使通过流水线的传输最大化。

定义 20 *f 的输入 M 的码字 $\chi_f(M)$ 是一组整数，对应于 M 的 f 等价类。*

得到一个码字 $\in \chi_f(M)$ 等同于得到 M $v_i(M)$ 的值，因为可以确定地将码字映射回 $v_i(M)$ 的等价类中的值。我们现在定义什么是 “计算 M 的码字”，我们将在证明中使用它：

定义 21 *如果我们可以计算 M 的码字 $\chi_f(M), \forall\ v_i(M) \in dom(M)$ 我们可以计算与 $v_i(M)$ 的等价类相关联的码字。*

在引理 2中，我们的码字给了我们一个 M 的传输要求的边界。

引理 2 *一张流表 t_i 只需要关于 M 的 $log_2(\lceil |dom(\mu_f(M))| - 1 \rceil)$ 比特的信息就可以正确执行 f。*

证明 1 *我们可以将任意 $v_i(M) \in dom(M)$ 编码为 $\chi_f(M)$ 中的码字，并且仍然可以传递足够的信息来计算 f。如果 $\mu_f(M)$ 可以取 $|dom(\mu_f(M))|$ 不同的值，我们可以从集合：$0, ..., |dom(\mu_f(M))| - 1$ 中为每个值分配一个唯一的码字，最多使用 $log_2(\lceil |dom(\mu_f(M))| - 1 \rceil)$ 位来表示。*

实现定理的证明： 基于我们对函数传输需求的描述，我们现在可以开始证明我们的实现定理。首先，我们将给出我们的关键潜在引理，引理 3，随后可以得到我们的实现定理。

引理 3 *如果 $\forall\ t_i \in \rho = \langle t_1, ..., t_n \rangle$ 有 $maxRules(t_i) > \tau_G(f)(\bar{M}_\rho(t_i))[dom]$，且 $2^{r(t_i)} > \tau_G(f)(\bar{M}_\rho(t_i))[ec]$，那么 $\forall t_i \in \rho = \langle t_1, ..., t_n \rangle$ 可以输出 $\chi_f(\bar{M}_\rho(t_i))$ 到 $r(t_i)$。*

给出引理 3后, 我们现在可以证明实现定理了。

证明 2 给定 f 和 p，我们会证明如果 $\kappa(pl) \trianglerighteq tau(f)$, $f \Longrightarrow pl$。考虑 $\kappa_\rho(\rho) \in \kappa(pl)$。

$\forall\ M \in \mathcal{M} : m_i \in M \rightarrow m_i \notin \bigcup_{t_i \in \rho} \bar{M}_\rho(t_i)$, $\kappa_\rho(\rho)(M) = (1,\ 1)$。因此，如果 $\kappa_\rho(\rho) > \tau_G(f) \Rightarrow$ 没有被 ρ 读取的所有 m_i 会被视作常量或者根本不会被 f 读取，因此 f 是从 $\bigcup_{t_i \in \rho} \bar{M}_\rho(t_i) \rightarrow \mathcal{R}$ 得到的映射。

进一步，给定 $\kappa_\rho(\rho) > \tau_G(f)\ \forall\ t_i \in \rho$, $maxRules(t_i) > \tau_G(f)(\bar{M}_\rho(t_i))[dom]$，且 $2^{r(t_i)} > \tau_G(f)(\bar{M}_\rho(t_i))[ec]$, 那么因此根据引理 *3* t_n 可以计算 $\chi_f(\bar{M}_\rho(t_n))$。

最后，考虑到如果一个 t_i 可以计算 $\chi_f(M_i)$，并且一个 f 是 $dom(M_i) \rightarrow \mathcal{R}$ 的映射, t_i 可以通过把 $\chi_f(M_i)$ 中的码字映射到它关联的 f 的输出的方式，计算 f 的输出 $\forall\ v_j(M_i) \in dom(M_i)$。

鉴于 t_n 是 ρ 的唯一输出, $\bar{M}_\rho(t_n) = \bigcup_{t_i \in \rho} \bar{M}_\rho(t_i)$。因此，$t_n$ 可以计算 f 的输出进一步，鉴于 t_n 是一个出口表，它总是可以将此输出传递回交换机。

因此，如果 $\kappa_\rho(\rho) > \tau_G(f), f \Longrightarrow \rho$。鉴于 $\kappa_\rho(\rho) \in k(p)$ 和 $\rho \in pl$, 我们已经证明如果 $\kappa(pl) \trianglerighteq \tau_G(f)$, $f \Longrightarrow pl$。

证明我们的实现定理所需的最后一步是证明 $\tau_G(f) > \tau(f)$ 所以才有 $\kappa(pl) \trianglerighteq \tau(f) \Rightarrow f \Longrightarrow pl$。下面引理 4给出了这一步骤的要点。

引理 4 M 等价类的数量被 $dom(\mu_f(M))$ 所限定。

证明 3 通过反证法，假设 $\exists\ (f,\ M) : |dom(M)/\sim_f| > dom(\mu_f(M)$。$v_i(M)$ 在 M 中的某个等价类的每个 $v_i(M)$ 必须生成一个 $v_i(\mu_f(M))$。根据鸽子洞原理[17]，如果 M 的等价类多于 $\mu_f(M)$, 则来自不同等价类的 M 的两个值必须生成相同的 $\mu_f(M)$。然而，根据引理 *1*, $\mu_f(M)$ 包含关于 M 的足够信息来确定 f 的输出值，因此 M 的这两个值必须在同一等价类中，这是一个矛盾。

推论 1 任一 M 的 f 等价类的数量 $G_f.vertexMinCut(M)$ 所限。

推论 2 特征方程 $\tau_G(f)$ 支配特征方程 $\tau(f)$。

因此，我们证明了我们的实现定理：$\kappa(pl) \trianglerighteq \tau_G(f) \Rightarrow f \Longrightarrow pl$。

无分支流水线实现定理的证明

上一小节中已经证明了我们的实现定理，$\kappa(pl) \trianglerighteq \tau_G(f) \Rightarrow f \Longrightarrow pl$，现在我们证明如果 pl 是无分支流水线，且 pl 的表容量足够大，并且每个匹配字段 $m_i \in \mathcal{M}$ 正好出现在 pl 的每一个表中，$\kappa(pl) \trianglerighteq \tau_G(f) \Leftrightarrow f \Longrightarrow pl$.

通过反证法，我们假设 $f \Longrightarrow pl$ 但存在 M 使得 $\kappa(pl)(M)[ec] < \tau_G(f)(M)[ec]$（我们省略 M 的域大小因为 pl 的表足够大）。因为 $f \Longrightarrow pl$ 且 pl 是一个无分支流水线且每一个匹配字段 $m_i \in \mathcal{M}$ 都正好出现在一个 pl 的流表中，因此我们有 $\kappa(pl)(M)[ec] \geq \prod \kappa(pl)(m_i)[ec]$（其中 $m_i \in M$）。因此我们有 $\kappa(pl)(M)[ec] \geq \tau_G(f)(M)[ec]$，而这与 $\kappa(pl)(M)[ec] < \tau_G(f)(M)[ec]$ 有矛盾。完成证明。

1.5 实验评估

我们现在对路由函数特征化的严密性（即计算出的值与真实值的差异性），路由函数特征化的时间复杂度，流水线特征化，以及流水线实现四个方面进行数值评估。所有的实验在 3.5 GHz Intel i7 处理器上运行，内存为 16 GB，系统为 Mac OSX 10.13。

1.5.1 路由函数特征化的严密性

通过比较以下一组路由函数的 τ 和 τ_G 的一个 M 等价类的数量（即 $ec(M)$），我们证明路由函数特征化的严密性。对于 τ 的 $ec(M)$ 计算将采用 f 等价类的定义进行计算。

```
\\  Routing function: simpleRoute
L1: def simpleRoute(Addr srcIP, Addr dstIP):
L2:   srcSw = hostTbl[srcIP]
L3:   dstSw = hostTbl[dstIP]
L4:   route = routeTbl[srcSw, dstSw]
L5:   return route
```

我们的第一个函数 `simpleRoute` 将包的 `srcIP` 和 `dstIP` 映射到它们的主机包交换机 `dstSw` 和 `srcSw`，然后查找它们之间的路由。

```
//  Routing Function: condRoute
L1: def condRoute(srcIP, dstIP):
L2:   srcSw = hostTbl[srcIP]
L3:   dstSw = hostTbl[dstIP]
L4:   routeCond = condTbl[srcIP, dstIP]
L5:   route = routeTbl[srcSw, dstSw, routeCond]
L6:   return route
```

我们的第二个函数 `condRoute` 通过引入一个路由条件变量来扩展 `simpleRoute`，该变量影响路由查找。

```
//  Routing Function: secureRoute
L1: def secureRoute(Addr srcIP, Addr dstIP):
L2:   if (isFiltered(srcIP)):
L3:     return Drop()
L4:   else:
L5:     route = fwdTbl[dstIP]
L6:     return route
```

我们的最后一个函数 `secureRoute` 将所有 `srcIP` 在一个过滤器列表中的数据包丢弃，并转发其余的数据包。

结果：结果见表 1.2。具体来说，在表 1.2中，第 2 列定义了 `srcIP`、`dstIP` 的域，第 3-6 列给出了每个表的输出范围 $O(tbls)$，第 7-10 列给出了每个函数的 $\tau(f)$ 和 $\tau_G(f)$ 中选定字段的值。

f	$bits(IP)$	O(hostTbl)	O(routeTbl)	O(condTbl)	O(fwdTbl)	τ(f)(srcIP)	τ(f)(dstIP)	τ_G(f)(srcIP)	τ_G(f)(dstIP)
smplR	10	100	2	N/A	N/A	100	100	100	100
smplR	10	100	30	N/A	N/A	100	100	100	100
smplR	10	100	5000	N/A	N/A	100	100	100	100
smplR	12	100	30	N/A	N/A	100	100	100	100
smplR	10	200	30	N/A	N/A	200	200	200	200
condR	10	100	30	50	N/A	1024	1024	1024	1024
condR	10	100	30	5	N/A	1024	1024	1024	1024
condR	10	100	30	1	N/A	100	100	1024	1024
scR	10	N/A	N/A	N/A	100	2	100	2	1024
b1(scR)	10	N/A	N/A	N/A	N/A	1	N/A	1	N/A
b2(scR)	10	N/A	N/A	N/A	100	1	100	1	100
onPkt	32	100	30	50	N/A	null	null	2^{32}	2^{32}

表 1.2: 不同统计量下路由函数的特征化结果

行 b1(scR) 和 b2(scR) 分别表示 secureRoute 的两个分支，L2→L3 和 L2→L4→L5→L6。当一个值不适用于给定函数时，我们记录 N/A，当计算失败时，$\tau(f)$ 的值为 null。

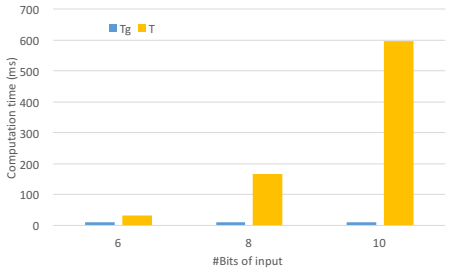
在我们对 simpleRoute 和 condRoute 的计算中，除 $O(\texttt{condTbl}) = 1$ 的特殊情况外，τ 和 τ_G 的结果在任何情况下都几乎相同。值得注意的是，我们的 $\tau(f)$ 和 $\tau_G(f)$ 值不受 $routeTbl$ O(routeTbl) 范围的影响。这说明当 $\texttt{hostTbl}(s_1)$ 不等于 $\texttt{hostTbl}(s_2)$ 时（s_1 和 s_2 是 $srcIP$ 的两个值），$s_1 \sim_f s_2$ 的概率非常小。

进一步发现，带有控制语句的函数：secureRoute 以及 onPkt，在 τ 和 τ_G 之间有一个很大的差距。这表明 τ_G 的界在有分支程序上是较松的。然而，正如行 b1(scR) 和行 b2(scR) 所示，我们可以对程序的每一个分支计算特征函数来解决这个问题。对于每一个分支（如 b1(scR) 和行 b2(scR) 所示），τ 和 τ_G 仍然保持着严密性。

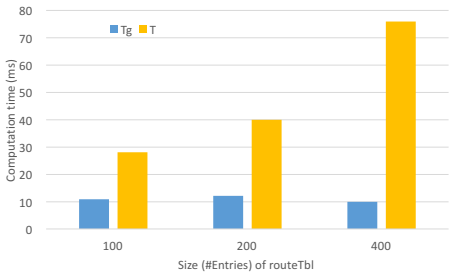
1.5.2 路由函数特征的时间复杂度

我们现在比较给定路由函数计算 τ 和 τ_G 所需的时间。如上一实验所述，τ 的计算将基于 f 等价类的定义。我们使用 simpleRoute 进行测试。其中 $O(\texttt{hostTbl}) = 100$，并改变输入范围（即输入比特位）和 routeTbl 的大小。

结果：图 1.6说明了随着输入以及 routeTbl 的变大，τ_G 的计算具有良好的可拓展性，即运算时间不改变。其中图 1.6a表示的是在固定 routeTbl 大小为 100 的情况，而图 1.6b表示的是固定输入外围为 6 的情况。从图 1.6a可以看出，随着输入比特位的变大，τ 的计算时间呈指数级增长而 τ_G 基本不变。从图 1.6b可以看出，随着 routeTbl 的变大，τ 的计算时间同样呈指数级增长而 τ_G 基本不变。这是由于 τ_G 的计算只依赖于 DFG 而 τ 需要对每个不同输入执行函数。



(a) 输入比特位长度变化



(b) 表大小变化

图 1.6: 评估结果

1.5.3 流水线的特征

我们现在评估流水线特征的计算时间和压缩性。其中压缩性表示特征函数的输出所占存储大小与 M 的全部可能所占存储大小的比值。我们考虑以下流水线进行评估:

1. **OF-DPA Abstract Switch 2.0:** OpenFlow 抽象数据平面抽象交换机 2.0（OF-DPA）是基于 OpenFlow 1.3.4 协议的抽象交换机模型，并在 OpenFlow 协议下允许对基于 Broadcom 的交换机进行编程。我们考虑了两个 OF-DPA 流表配置：（1）桥接和路由（BR）；（2）数据中心覆盖隧道（OT），它们分别有 5 个阶段（7 个流表）和 3 个阶段（3 个表）。[2]

2. **PicOS:** PicOS 是一个应用于白盒交换机的网络操作系统。它提供了跨 HP、Edgecore 和 Pica 交换机的可编程性。我们考虑 PicOS 提供的两个固定流水线：（1）IP 路由流水线（IPR）；（2）策略路由流水线（PR），分别包含 4 个和 5 个表对应在 4 和 5 个阶段。[18]

结果：表1.3是我们对评估流水线的特征结果。其中，我们说在一个流水线 pl 中 M 有效，所指的是 $\kappa_\rho(\rho)(M)$ 的值可以通过 $\kappa_\rho(\rho)(M)$ 定义的第一个公式算出。结果表明，尽管在理论上 $\mathcal{M}$ 子集的数量巨大，但实际上有效 M 并不多。同时流水线特征的计算也非常小。

流水线	# 路径数量	时间 (ms)	# 有效 M	$\#M$
`ExampleDP`	3	8	6	22
PicOS BR	4	13	19	$3 * (2^{24}) + 2^7$
PicOS OT	2	7	5	$2^{24} + 16$
Broadcom IPR	1	7	4	2^7
Broadcom PR	3	9	14	$2 * (2^{24}) + 2^7$

表 1.3: 流水线的特征结果。

1.5.4 流水线实现的分析

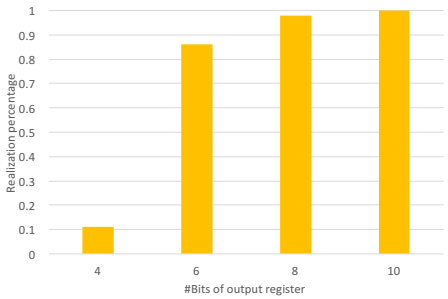
我们现在评估在流水线中成功实现函数的百分比，以查看成功实现的因素。我们考虑具有不同表内容的示例函数 `onPkt`。具体来说，对于 `condTbl` 和 `hostTbl`，我们随机设置表的输出值，范围从 10 到 30。这意味着 `dstCond` 和 `dstSW` 的域大小是从 10 到 30（即平均域大小为 20）。对于流水线方面，我们随机决定表的数量（从 2 到 4）以及每个表的输出寄存器的位长（从 4 到 10）。另外我们设置生成的流水线的匹配字段必须包含 `onPkt` 所需的五个匹配字段，并且每个匹配字段只能出现在一个表中。最后我们通过实现定理计算 `onPkt` 的成功实现百分比和生成的流水线。

结果：如图 1.7所示。其中，图1.7a考虑的具有两个表的流水线；图1.7b为 3 个表的流水线；图1.7c为 4 个表。对于每种情况，我们变化每个表的寄存器位长度，并计算实现成功的百分比。从图 1.7a的结果中，我们可以发现具有更多位寄存器的流水线可以实现更高百分比的功能。但是，当长度大于一定阈值时，成功实现百分比不会增加太多。阈值由函数中变量域的大小确定。如图1.7a所示，实现成功的百分比在 4 位和 6 位寄存器长度之间的差距（即等效类的可用大小从 16 到 64）可以通过变量的平均域大小（即 20）在 16 到 64 之间的范围来解释。

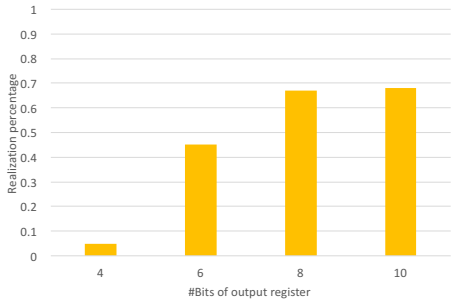
进一步，我们可以发现流水线的结构也决定了实现成功的百分比。具有 2 个表的流水线（即没有分支）具有相对较高的成功实现百分比，而具有 3 或 4 个可能包含分支的表的流水线具有较低的实现百分比，因为这些流水线的结构可能并不好（即匹配字段和表之间的随机映射）。

1.6 本章小结

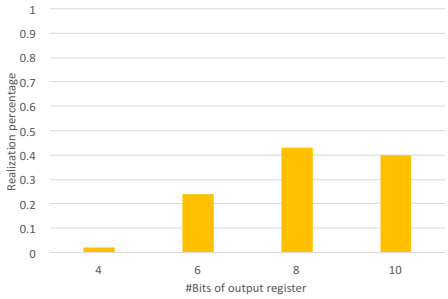
本章我们针对高级 SDN 程序对于固定结构数据通路的实现问题，提出了将高级 SDN 程序和底层数据通路统一的特征空间。并将实现问题转化为空间中的比较问题，并给出流水线实现定理以及相关证明。实验表明，虽然我们的流水线实现定理给出的是充分条件，但其严密性较好，在实际情况中不会太大误差。



(a) 表数量 = 2



(b) 表数量 = 3



(c) 表数量 = 4

图 1.7: 对于不同表数量和寄存器位长度的实现成功百分比。

参考文献

[1] FOUNDATION O N. OpenFlow Switch Specification 1.3.0[EB]. 2014. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf.

[2] OpenFlow Data Plane Abstraction (OF-DPA): Abstract Switch Specification Version 2.0[EB]. Broadcom. 2014 [2017-06-03]. www.broadcom.com.

[3] BOSSHART P, DALY D, GIBB G, et al. P4: Programming Protocol-independent Packet Processors[J]. SIGCOMM Comput. Commun. Rev.'14,

[4] SIVARAMAN A, CHEUNG A, BUDIU M, et al. Packet transactions: High-level programming for line-rate switches[C]//Proceedings of the 2016 ACM SIGCOMM Conference. 2016: 15-28.

[5] BOSSHART P, GIBB G, KIM H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[C]//SIGCOMM'14.

[6] FOSTER N, HARRISON R, FREEDMAN M J, et al. Frenetic: A network programming language[J]. ACM Sigplan Notices, 2011, 46(9): 279-291.

[7] VOELLMY A, WANG J, YANG Y R, et al. Maple: Simplifying SDN programming using algorithmic policies[J]. SIGCOMM'13,

[8] ARASHLOO M T, KORAL Y, GREENBERG M, et al. SNAP: Stateful network-wide abstractions for packet processing[C]//SIGCOMM'16.

[9] HINRICHS T L, GUDE N S, CASADO M, et al. Practical declarative network management[C]//Proceedings of the 1st ACM workshop on Research on enterprise networking.

[10] NELSON T, FERGUSON A D, SCHEER M J G, et al. Tierless Programming and Reasoning for Software-defined Networks[C]//Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. Seattle, WA: USENIX Association, 2014: 519-531.

[11] REICH J, MONSANTO C, FOSTER N, et al. Modular sdn programming with pyretic[J]. Technical Reprot of USENIX, 2013.

[12] SHAHBAZ M, CHOI S, PFAFF B, et al. Pisces: A programmable, protocol-independent software switch[C]//Proceedings of the 2016 ACM SIGCOMM Conference. 2016: 525-538.

[13] SCHLESINGER C, GREENBERG M, WALKER D. Concurrent NetCore: From policies to pipelines[C]//ACM SIGPLAN Notices:vol. 499. 2014: 11-24.

[14] JOSE L, YAN L, VARGHESE G, et al. Compiling Packet Programs to Reconfigurable Switches[C]//12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15). Oakland, CA: USENIX Association, 2015: 103-115.

[15] SUN X, NG T E, WANG G. Software-Defined Flow Table Pipeline[C]//Cloud Engineering (IC2E), 2015 IEEE International Conference on. 2015: 335-340.

[16] PAN H, GUAN H, LIU J, et al. The FlowAdapter: Enable flexible multi-table processing on legacy hardware[C]//Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. 2013: 85-90.

[17] AJTAI M. The complexity of the pigeonhole principle[C]//[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science. 1988: 346-355.

[18] Scaling up SDNs using TTPs (Table Type Patterns)[EB]. Pica 8. 2015 [2017-06-03]. www.pica8.com.