应用于 Web 服务器匹配算法的 FPGA 实现

孟旭东[1,2],许强凯[1,2]
(1. 南京邮电大学 宽带无线通信与传感网技术教育部重点实验室,江苏 南京 210003;
2. 南京邮电大学 江苏省电信网络融合实验室,江苏 南京 210003)

摘 要:Web 服务已经成为现代人网络生活的一部分,人们需要通过 Web 迅速地获取信息,需要在 Web 上快速地搜索关键字。在 Web 服务器端实现快速搜索,需要 Web 服务器能够快速地对流经服务器的数据流进行字符串匹配。对字符串匹配算法进行了系统介绍,其中重点分析了利用位并行计算的 Shift-Or 算法。之所以利用 FPGA 来实现,是因为 FPGA 的实现方式在速率上高于软件实现方式,在灵活性上高于专用集成电路的实现方式。在 FPGA 上实现了 Shift-Or 字符串匹配算法,并在千兆以太网的环境下进行了实验测试。实验结果表明,该方法能够满足在高速网络环境下对数据包内容的深度检测。

关键词:Web 服务器;字符串匹配;Shift-Or;FPGA
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2016)12-0142-06
doi:10.3969/j.issn.1673-629X.2016.12.031

Implementation of FPGA Applied to Web Server Matching Algorithm

MENG Xu-dong[1,2], XU Qiang-kai[1,2]
(1. Key Laboratory of Broadband Wireless Communication and Sensor Networkof Ministry of Education, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. Telecommunications Network Integration Lab of Jiangsu, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: Web services have become part of the modern life, and people need to get information through the Web quickly and require fast keyword search on the Web. To realize fast pattern matching on the Web server side, the Web server is needed to process the data stream through the server for string matching quickly. String matching algorithm is introduced systematically, in which the analysis is mainly focused on the use of a Shift-Or algorithm with parallel computing. Using FPGA to implement, because FPGA-based implementation can have a higher process rate than software implementations, and be more flexible than the ASIC implementation. The Shift-Or string matching algorithm is implemented in FPGA, and then tested in gigabit Ethernet. The results show that the design can meet the high speed packets rate under network environment of gigabit Ethernet.

Key words: Web server; string matching; Shift-Or; FPGA

0 引 言

在网络这个数据的海洋里,数据流量正不断增长,人们迫切需要对数据进行搜索,需要在 Web 服务器[1]上实现字符串匹配,工作量巨大。这一要求对数据搜索查找技术提出了巨大挑战。字符串匹配是其核心技术。在网络服务器[2]端,数据匹配的需求几乎无处不在,尤其是在网络安全领域。网络安全领域中的一系列应用,例如防火墙、入侵检测防护、垃圾邮件过滤、深度内容检测等,都与字符串匹配紧密相关[3]。字符串匹配在上述安全领域的应用中都处于核心位置,并且占用着大量的计算资源。举个例子,在检测入侵的网络安全应用中,字符串匹配是系统能否成功发现包含安全威胁信息的关键。入侵检测系统先把部分已知具有安全威胁特征的模式串保存下来,定义成一系列的规则。当有数据流入内部网络时,系统按照这些预先定义的规则和预先存下来的模式对流入数据进行对

收稿日期:2016-02-15 **修回日期**:2016-06-09 **网络出版时间**:2016-11-21
基金项目:国家“973”重点基础研究发展计划项目(2013CB329005)
作者简介:孟旭东(1959-),男,副研究员,研究方向为电信网络和 IP 网络的交换、异构网络集成及业务融合、未来互联网体系结构、网络计算与分布式处理;许强凯(1991-),男,硕士研究生,研究方向为下一代通信网络。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20161121.1641.036.html

比,检查出数据包内容中可能存在安全威胁的信息。据统计,在这样的安全系统里,字符串匹配所消耗的计算资源占到了系统所有计算资源的 50% 以上[4]。字符串匹配本身需要消耗大量的计算资源,这一点对在网络服务器端实现字符串匹配提出了较高要求。

在目前已有的字符串匹配算法中,最直接的方法是 BF 算法,俗称蛮力算法。也就是不涉及任何技巧,逐个字符进行比较判断,从而得到两个字符串是否相同的结论。除了 BF 算法之外,研究人员提出了各种各样的字符串匹配算法。这些算法中,有非常经典的 Knuth – Morris – Pratt (KMP) 算法[5]、Boyer – Moore (BM)算法[6];也有一些新式算法采用较新技术思想(例如位并行),其中的代表是 Shift–And/Shift–Or 算法,它通过同时完成多个位的计算达到同时处理多个字符串位置的目的,从而大幅提高了匹配效率[7]。

在网络服务器这样的对信息处理速度要求非常高的地方,传统的利用通用处理器 CPU,通过软件实现字符串匹配存在运行速率不足的问题。软件实现方式不能保证线速处理,从而造成网络时延较大或信息检索不充分。由于通用处理器性能的提升跟不上网络速率的提升,一般在通用处理器上以软件方式实现字符串匹配只能用于低于 100 Mbps 的低速网络中。所以利用硬件来实现字符串匹配算法。

1　Shift–Or 算法的 FPGA 实现设计概要

Shift–Or 算法[8]利用位并行的方法提高了字符串匹配速率,通过一个位掩码 D,表示模式串前缀和文本串后缀的匹配情况。它同样是先根据模式字符串构造一个表 B,用来记录字母表中每个字符的位掩码 $b_m\cdots b_1$。如果 $p_j = c$,掩码 $B[c]$ 的第 j 位被置 0,其余为 1。这里的位掩码 D 初始为全 1。若模式串前缀 $p_1\cdots p_j$ 是文本串 $t_1\cdots t_i$ 的后缀,就把位掩码 $D = d_m\cdots d_1$ 的第 j 位置 0,并称 D 的第 j 位是活动的。

当模式串 $P = p_1\cdots p_m$ 长为 m,d_m 是活动的时候,就说明有一个成功匹配。每次读入下一个文本字符 t_{i+1} 时,需要重新计算位掩码 D。同样是利用了位并行的方式,使得 D 的计算可以在常数时间内完成。D 的更新公式如下:

$$D = (D \ll 1) \mid B[t_{i+1}]$$

可以发现 Shift–Or 算法实际上是 Shift–And 算法的一种富技巧性的实现。它通过对位取反从而去掉 $0^{m-1}1$,从而简化了计算,提高了速度。Shift–Or 算法的伪代码如下[9]:

Shift – Or($p = p_1\cdots p_m, T = t_1\cdots t_n$)
Preprocessing
For $c \in \Sigma$ Do $B[c] = 1^m$
For $j \in 1\cdots m$ Do $B[p_j] = B[p_j]\&1^{m-j}01^{j-1}$
Searching
$D = 1^m$
For $i \in 1\cdots n$ Do $D = (D \ll 1) \mid B[t_i]$
if $D \mid 01^{m-1} \neq 1^m$ Then find an occurrence at $i - m + 1$
End of For

假设在进行信息查找之前,FPGA 内没有事先存储需要匹配的模式字符串。根据 Shift–Or 算法,FPGA 需要在对文本字符串进行信息搜索之前完成对模式字符串的预处理,计算得到算法需要的表格 B 并存在 RAM 内。先向 FPGA 发送包含模式字符串的数据包,让 FPGA 对模式串进行预处理。预处理完成后,FPGA 开始等待接收包含文本字符串的数据包,对接收到的文本串计算字符串匹配结果。

2　利用该字符串匹配模块工作的系统设计

该系统工作在网络的数据链路层[10],利用千兆以太网口收发以太网帧[11]。但它对数据的处理不局限于以太网数据帧的帧头部信息。以太网帧中封装了数据包,数据包的净荷里存放着接下来所做的字符串匹配所需要的模式字符串和文本字符串。系统在接收到一个数据帧后,会传给 FPGA 芯片,可以对帧内所有数据进行操作处理。

以太网帧中包括 6 字节的目标 MAC 地址,6 字节的源 MAC 地址,2 字节的数据类型,4 字节的校验和,以及帧中间的数据包部分。以太网协议规定数据包最少 46 字节,最长 1 500 字节。而这里除数据包以外都可以认为是帧的头部信息,属于数据链路层,不影响在应用层做字符串匹配。FPGA 在接收了这样一个数据帧后,首先是存到 RAM[12]里,再进一步对数据进行处理。协议规定了以太网帧的固定结构,每一个以太网帧的头部信息长度是固定的,可以利用这一点,准确找到在 FPGA 中的 RAM 里存下来的数据帧内部的数据。

以太网帧封装的是网络层的数据包,也就是通常所说的 IP 包。IP 包里包含着用户数据,但是和上述的帧结构类似,IP 包存在一个 20 字节的头部信息。IP 报文里的头部包括源 IP 地址、目标 IP 地址、头校验和等信息,属于网络层信息。IP 数据包里接下来是 TCP/UDP 包,同样也包含固定长度的头部信息。对 UDP 包而言,UDP 报头包括 2 字节的源端口号,2 字节的目的端口号,2 字节的 UDP 长度和 2 字节的校验和,一共 8 字节的头部信息。上述所有的头部信息,因为都是固定长度,所以在 RAM 中的存储相对位置也是固定的。利用信息在 RAM 中相对位置固定的特点,可以方便地找到处理所需要的数据。

按照上述原理,把应用层的模式字符串和文本字

符串分别封装在不同的数据包中,从外部通过以太网帧的形式传入系统,利用 UDP 的端口号来对其进行区分,以完成不同的处理。根据算法工作流程(见第三节),如果判定为模式串,根据 UDP 报头里的数据包长度确定模式串的长度,并在 FPGA 中对模式串进行 Shift-Or 算法预处理;如果判定为文本串,还是可以根据 UDP 报头里的数据包长度确定文本串的长度,并在 FPGA 中对文本串进行 Shift-Or 算法匹配,这两个过程可以分开并行执行。如果匹配成功,可以将匹配成功信息通过数据包的形式返回给外部接口。系统的结构如图 1 所示,系统核心部分在字符串匹配模块。

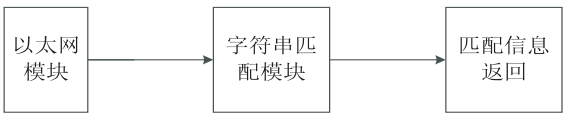

图 1 系统结构

2.1 字符串匹配模块的具体实现

Shift-Or 算法及其在 FPGA 上实现的思路已经在前面进行了详细分析。字符串匹配模块下还可以更细致地分为模式串预处理模块和文本串匹配模块。

字符串匹配所需的模式串和文本串分别由具有不同端口号的 TCP/UDP 包封装起来。FPGA 可以根据所收到的数据包中的 TCP/UDP 端口号来判断已经存入 RAM 的数据包中的净荷是模式串还是文本串。在 verilog 程序中,模式串预处理模块和文本串匹配模块分别用一段 always 语句块实现。

2.1.1 模式串预处理实现

如果 FPGA 判定正在处理的数据包所含净荷是模式字符串,Shift-Or 算法要求对模式字符字符串进行预处理。在这个预处理中,算法会根据读入的模式字符串制作一张模式特征表格。模式特征表 B 的每一行对应模式串中的一个字符。定义的表格 B 有 27 行,可以处理 26 个英文字母,其他的任意字符统一记为‘*’。

在 ASCII 编码中,一个英文字符的长为 1 字节,即 8 位比特位,并且有大小写之分,即大写的字符‘A’和小写的字符‘a’编码不同。通过对字符编码的判断,实现了忽略英文字符大小写的模式匹配。例如,字符串“CHINA”在该字符串匹配系统中和字符串“china”效果相同。

字符‘A’的 ASCII 编码为 65,即 8'b0100_0001,字符‘a’的 ASCII 编码为 97,即 8'b0110_0001,在计算模式特征表格 B 时,同时考虑了这两种情况,字母‘A’或‘a’在模式串中的特征(在模式串中出现的位置)都在表格 B 的第一个行向量 $B[0]$ 中表示出来。同理,字符‘B’的 ASCII 编码为 66,即 8'b0100_0010,字符‘b’的 ASCII 编码为 98,即 8'b0110_0010,在计算模式特征表 B 时,字母‘B’或‘b’在模式串中的特征都在表格 B 的第二个行向量 $B[1]$ 中表示出来。对于非英文字母表的字符,一概认为是字符‘*’,相应的在表格 B 中对应的行向量 $B[26]$ 就为全 1。

在计算表 B 时,需要考虑两个位置问题:

(1)目前正在计算模式串的第几个字符位置;

(2)计算的位向量应该是表格 B 中的第几行。

变量 decoder 就是为了描述第一个位置问题而设置的。当目前正在处理的字符在模式串的第一位时,对模式特征表 B 的计算结果只针对表的某一行向量的最低位(即最右边的一位),具体是哪一行,那是第二个位置问题。在顺序读入模式字符串时,应该一直明确当前读入的字符在模式字符串中的位置。为了明确这一位置,就另行设定了这个 decoder 向量。当正在读入的字符是模式串的第一个字符时,相应的 decoder 的最低一位(最右边一位)就设置为 0,其他位设置为 1。同理,当正在读入的字符是模式串的第二个字符时,相应的 decoder 从右往左的第二位就设置为 0,其他位设置为 1。其他情形以此类推。

如果模式串长为 16,就可以把算法中的字符 c 对应的模式特征表 B 中行向量 $B[c]$ 的长度设定为 16,即为一个 reg[15:0]的变量。在计算表 B 时,另外设定了一个长为 16 位的寄存器变量 decoder,变量类型为 reg[15:0],用来记录字符 c 在模式串从低到高(0 到 15 位)16 个可能的位置中出现的确切位置,从而方便计算 $B[c]$。计算 decoder 的部分 verilog 代码如下:

```
case(str_len)
5'd0:decoder[15:0]<=16'hfffe;
5'd1:decoder[15:0]<=16'hfffd;
5'd2:decoder[15:0]<=16'hfffb;
5'd3:decoder[15:0]<=16'hfff7;
//这里省略 5'd4 到 5'd15 的情形
endcase
```

如果读入的字符是模式字符串的第一个字符,decoder 的二进制表示就是 16'b1111_1111_1111_1110,即为 16'hfffe。同理,当处理到模式串的第二个字符时,变量 decoder 的二进制表示就是 16'b1111_1111_1111_1101,即为 16'hfffd。这样,就通过设置一个类似位掩码向量的 decoder 寄存器变量,解决了上述的第一个位置问题。

第二个位置问题需要由正在处理的模式串字符确定。事先规定模式特征表 B 的行数为 27,前 26 行表示 26 个英文字母,而且不区分大小写,最后一行表示其他任意字符。

还是在模式串长度不超过 16 个字符的假设下,进行模式串特征制表。按照 Shift-Or 算法,每读入一个模式串字符 c,都可能需要更新模式特征向量 $B[c]$。

在这里需要用到上述第一个位置问题里的寄存器变量 reg[15:0] decoder,先把当前的 decoder 向量和与当前读入的模式串字符 c 相应的特征向量 B[c]作按位与运算,这样就可以针对 B[c]的某一位进行操作,继而完成对 B[c]的赋值。部分 verilog 代码如下:

```
case(c)
8'b0100_0001: B[0]<=(B[0] & decoder[15:0]);//字符'A'
8'b0110_0001: B[0]<=(B[0] & decoder[15:0]);//字符'a'
8'b0100_0010: B[1]<=(B[1] & decoder[15:0]);//字符'B'
8'b0110_0010: B[1]<=(B[1] & decoder[15:0]);//字符'b'
//这里省略另外 24 个英文字母
default: B[26]<=(B[26] & decoder[15:0]);//其他字符
endcase
```

可以发现 Shift-Or 算法实际上利用了空间换时间[13]的算法思想,在开始对文本串进行字符串匹配前,需要对模式串进行分析计算,得出一张模式字符串特征表,也就是算法的预处理过程。构成这张二维数据表的元素为 0 和 1,表格包括了字符串匹配系统所有可以识别的字符。这里将系统可以识别的字符集定义为所有英文字母,所以系统的模式特征表格有 27 行,前 26 行分别对应 26 个英文字母,最后一行代表可能在文本串出现的任意其他字符,比如空格或标点符号。每一行作为描述对应字符在模式串中的特征,反映了该字符在模式串中的出现位置信息。在接下来对文本串进行匹配时,就通过这样一张模式字符串特征表来对每个读入的文本串字符操作,用表格 B 里的行向量和字符串匹配状态编码,即位掩码 D 进行位并行操作,节省了大量时间。在制表时考虑到字母大小写的问题,从而实现了允许大小写通用的字符串匹配,增加了系统的灵活性。

2.1.2　文本串匹配模块的实现

字符串匹配模块对从 RAM 取出的数据进行判别后分流,对模式字符串和文本字符串分别进行不同的处理。文本字符串实际上应该是在 FPGA 对模式字符串已经预处理过后才送入 FPGA 进行字符串匹配,所以在实现文本串匹配模块时,假设模式字符串已经在之前的数据包中给入 FPGA,并且算法对模式串的预处理部分已经完成,即模式串特征表已经计算得出。

Shift-Or 算法在对文本串进行匹配时,借助一个由 0、1 组成的位掩码 D,判断匹配是否成功。这个位掩码 D 在之前介绍过,长度不少于模式字符串长,和预处理模块计算得到的模式特征向量 B[c]的长度相同。同时,D 也可以理解为匹配状态编码。位掩码 D 的初始值为全 1,当模式字符串的第一个字符和最后读入的文本串字符相匹配时,D 的最低位由 1 改为 0,其余位为 1;当模式串的前两个字符和最后读入的文本串的两个字符同时匹配时,D 的第二位由 1 改为 0,其余位为 1,以此类推。这样的状态编码清晰记录和表现了字符串匹配过程。可以非常直观地推断,如果模式串长为 m,那么每当位掩码 D 的第 m 位(从右往左第 m 位)为 0 时,表明找到了一个成功匹配。

与之前一样,这里还是先假设模式串的长度为 16。那么,模式特征表 B 中字符 c 对应的 B[c]特征向量长度就为 16,字符串匹配过程中的状态编码 D 也为长度 16 的向量。在设计中,模式特征向量 B[c]和字符串匹配状态编码 D 都定义为 reg[15:0]的寄存器变量。负责更新匹配状态编码 D 的部分 verilog 代码如下:

```
case(ch)
8'b0100_0001: D[15:0]<=(D[15:0]<<1 | B[0]);//'A'
8'b0110_0001: D[15:0]<=(D[15:0]<<1 | B[0]);//'a'
8'b0100_0010: D[15:0]<=(D[15:0]<<1 | B[1]);//'B'
8'b0110_0010: D[15:0]<=(D[15:0]<<1 | B[1]);//'b'
//这里省略读入字符从'C'/'c'到'Z'/'z'的情况
default: D[15:0]<=(D[15:0]<<1 | B[26]);//'*'
endcase
```

包含有文本串的数据包流进 FPGA 被顺序读取,FPGA 每读取一个文本字符,对字符进行判断查表处理。如果该文本字符属于英文字母字符集,则系统可以在模式特征表格 B 里找到相对应的模式特征向量,更新当前的字符串匹配状态编码;如果不属于英文字母字符集,则可以肯定与模式串不匹配,所用的 B[*]为全 1,更新得到的匹配状态编码会直接回到初始全 1 状态。字符串匹配是否成功可以从这个匹配状态编码得出结论。

2.2　匹配信息返回模块实现

在 Shift-Or 算法中,字符串匹配是否成功是根据匹配状态编码,即文本串匹配模块中的位掩码 D 的最高位来判断。如果 D 的最高位为 1,则匹配不成功或还没有完全匹配;如果 D 的最高位为 0,则模式串在文本串中匹配成功。确定匹配成功后,系统需要以某种方式返回这个匹配成功信息,其中还应该包括在文本串中的成功匹配位置。

每读入一个文本串字符 c,Shift-Or 算法会更新一次匹配状态编码 D。延续之前的例子,在 FPGA 实现中,verilog 代码为:

```
D[15:0]<=(D[15:0]<<1 | B[c]);
```

注意这里的赋值采用的是非阻塞赋值符,在一个语句块内,首先计算该块内所有非阻塞赋值符右边的

值,然后这些值再赋给左边的寄存器,赋值过程分成了两个小步骤。在这里对状态编码 D 的更新中,新的 D 要在下一时钟周期内才稳定。在完成对新的匹配状态编码计算后的下一个时钟周期内检查这个新的状态编码 D 的最高位,为 0 则匹配成功。

在顺序读入文本串的过程中,设置一个计数器用来计算当前正在处理的字符在文本串的序数。当计数器值为 i 的时候,匹配状态编码 D 的最高位变成 0,表明在搜索到文本串的第 i 个字符的时候,模式串和已经读入的文本串后缀完全匹配。如果模式串长度为 m ,可推算出字符串匹配成功发生在文本串第 $i-m+1$ 个字符的位置。这个匹配成功的位置信息作为数据净荷封装在一个数据帧里,通过以太网模块返回。

3 硬件与系统测试

3.1 硬件开发平台

实验所使用的硬件开发平台包括两个主要模块:千兆以太网模块和 FPGA 开发板模块。以太网模块提供了一个千兆以太网口,用来收发数据帧。与以太网模块相连的 FPGA 开发平台是系统设计的重点,所要完成的字符串匹配算法在这里实现。

千兆以太网模块使用了 Marvell 公司的 88E1111 芯片。88E1111 是一个用于吉比特以太网的物理层收发器,它支持 1000BASE-T、100BASE-T 和 10BASE-T 类型的以太网。88E1111 芯片使用标准数字 CMOS 工艺制造,包含所有必需的有源电路来实现物理层功能。

所选用的 FPGA 开发板型号是 EP4CE40F23C8N,包括 Altera 公司的 Cyclone IV 可编程 FPGA 芯片。开发板可以提供高达 3.125 Gbps 的数据传输速率,支持千兆以太网协议,并且功耗低,非常适合于高速网络数据处理的应用。

3.2 实验环境搭建

如图 2 所示,实验包括两台电脑和千兆以太网 FPGA 平台。

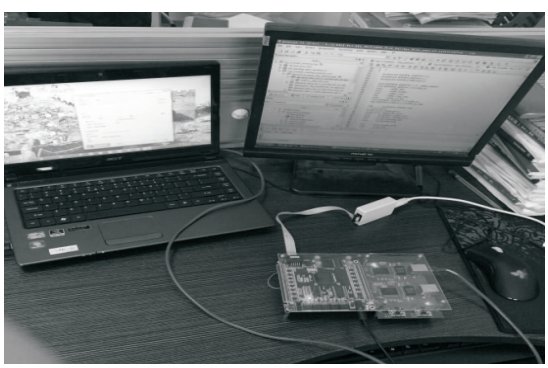

图 2 实验设备

图中左边计算机和以太网模块由网线连接,右边计算机与 FPGA 模块由 USB blaster 连接。通过左边计算机上的科来数据包产生软件产生以太网数据帧,与以太网模块通信。在右边计算机上的 Quartus II 软件上使用 verilog 语言编程,然后通过 USB blaster 将程序下载进 FPGA。FPGA 开发板型号为 EP4CE4DF23CBN,包括 Altera 公司的 Cyclone IV FPGA 芯片。千兆以太网模块使用 Marvell 公司的 88E1111 芯片。FPGA 使用锁相环 PLL[14] 产生频率为 125 MHz 的时钟,每时钟处理一个字节,而每字节 8 bit,与千兆以太网口速率匹配。

3.3 测试结果及分析

测试中使用科来数据包产生软件产生包含模式串的数据,经过以太网模块发送给 FPGA 模块。通过网络抓包软件获取 UDP 数据包。这个 UDP 数据包的目标端口号是 0x1000,系统中表示净荷中的数据为模式字符串。测试使用的模式字符串为“ABCABCABCABCABCA”,长度为 16。

FPGA 在接收到包含模式字符串的数据之后,将净荷数据取出按照 Shift-Or 算法对模式串进行预处理,主要任务就是计算模式特征表 B。通过软件 Quartus II 中的 SignalTap 可以观察到算法运行情况。如图 3 所示,可以看到定义的寄存器变量 decoder 和读取的模式串字符 q1 的时序图。变量 clk 为时钟。str_len 用来对模式串字符计数。字符串“A,B,C”的 16 进制表示就是“41h,42h,43h”,从图中可以找到对应的位置。模式串预处理模块中 decoder 计算完成后开始计算模式特征表 B,在模式串预处理时序图中最下一行的 b 变量可以观察到。

模式串预处理完成后,向 FPGA 发送文本字符串,与之前发送模式字符串的方法相同。文本串数据包与模式串数据包不同的是 UDP 的目标端口号,包含文本串数据的 UDP 使用 0x8000 的目标端口号。用抓包软件捕捉到 UDP 包。

FPGA 接收到上述包含文本串的数据包后,对其展开深度内容检测,即对 UDP 数据净荷进行模式串匹配。在 SignalTap 中可以观察到在时钟信号 clk 的驱动下,文本串顺序读入字符 q2 和匹配状态编码 D 的时序图。这里的字符串匹配用到了模式特征向量 $B[0]$, $B[1]$, $B[2]$,它们分别为 6DB6h,DB6Dh,B6DBh。

为了更细致地观察匹配状态编码 D ,将字符串匹配成功的部分放大,如图 4 所示。在编号 35 的时钟周期内,16 位状态编码 D 的最高位变位 0,表示匹配成功。

注意到当 D 的最高位为 0 时, D =“0110110110110110”,以“011”的形式重复,这正好符合测试使用的模式串“ABCABCABCABCABCA”以“ABC”的形式重复。通过手工计算,也可以验证该算法在 FPGA 上

正确运行。实际上在图 4 中编号 33 的时钟信号上升沿的地方,模式字符串就已经完整地在文本字符串中出现了,而此时的 D 还没有更新,要到下一个时钟周期内 D 才会更新,最高位改写为 0。

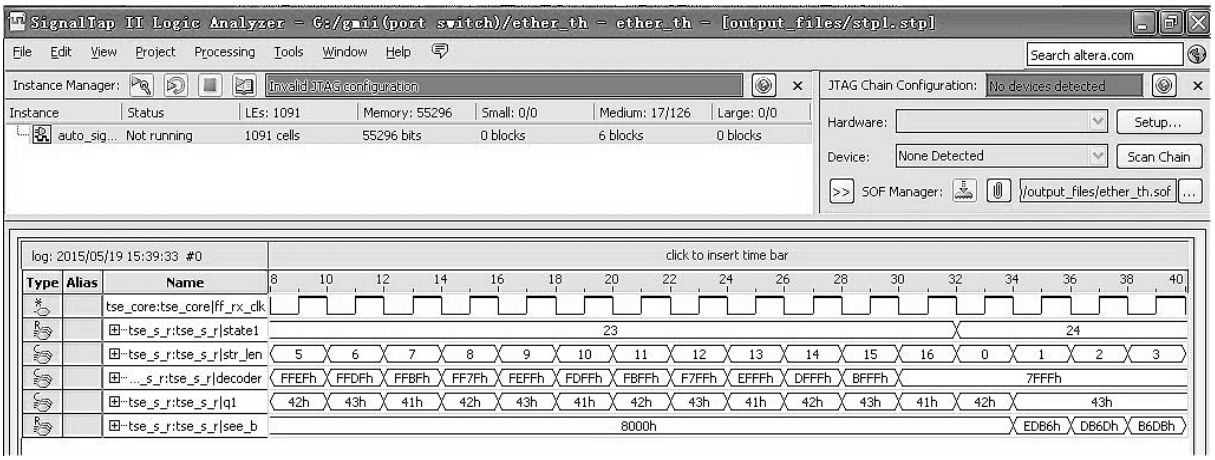

图 3　模式串预处理时序

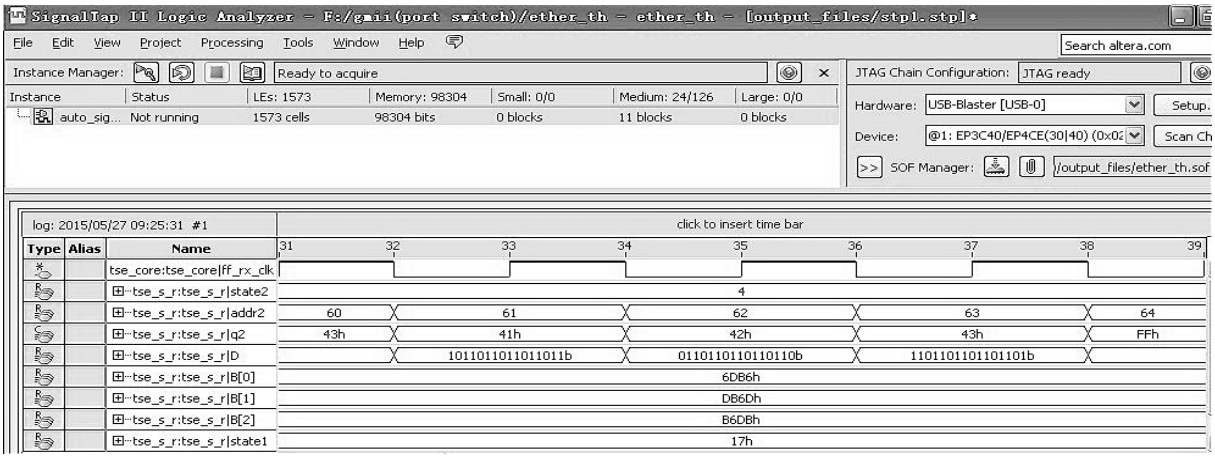

图 4　字符串匹配结果时序

算法理论上,模式串“ABCABCABCABCABCA”对应的模式特征表项 $B[0]$ 代表字符‘A’,$B[0]$ 应该为“0110110110110110”。实际测试中,计算得到 $B[0]$ 是“6DB6”(十六进制表示),换算成二进制为 0110_1101_1011_0110,与理论值相同。同样的方式可以验证字符‘B’的模式特征向量 $B[1]$ 和字符‘C’的模式特征向量 $B[2]$,实际的模式特征表与理论相符。

当顺序读入文本串时,匹配状态编码 D 会相应更新,可以跟踪 D 的更新过程来验证系统的工作。初始时,16 位的 D 为全 1,实验中,读入不属于英文字符集的字符时,D 一直保持全 1。当读入字符‘A’,D 在下一时钟周期里更新为“1111_1111_1111_1110”,继续读入字符‘B’时,D 更新为“1111_1111_1111_1101”;读入文本串的后缀为“ABCA”时,D 将在下一时钟周期内更新为“1111_1111_1111_0110”。可以发现实验中匹配状态编码 D 的更新与 Shift-Or 算法理论完全相符。

系统的字符串匹配处理速率与系统时钟频率有关。该系统 FPGA 通过锁相环 PLL 产生频率为 125 MHz 的时钟信号。系统每个时钟周期处理一个字符,而一个字符长 8 bit,可以算出该系统可以支持千兆以太网上的字符串匹配应用。

4　结束语

网络服务器承载了大量的用户数据流量,在当今网络数据的传输速率飞速发展,并且用户越来越多的背景下,数据流量很容易达到千兆甚至万兆比特每秒。数据传输速度的提高已经成为了用户最基本、最重要的需求。然而随着计算机通信技术的进步,如果只是简单地传输数据,那么并不能实现更有意义的功能,需要对服务器中所承载的数据进行处理。文中在千兆以太网的测试环境中,实现了线速对数据包净荷内容的字符串匹配。选择 FPGA 作为字符串匹配算法实现平台,针对 FPGA 在并行计算上的强大性能以及 FPGA 可编程的灵活性,选用以位并行计算为特点的 Shift-Or 算法。实验在千兆以太网 FPGA 开发平台上进行,在 Altera 生产的 Cyclone IV FPGA 芯片里实现了 Shift-

(下转第 152 页)

根据实际测试结果可知,文中所提出的算法适用于基于北斗或者其他实时定位技术的全自动判定及报站,能够较好地解决公交车自动上下行判断的问题,且在公交车临时调头的情况下能够实现报站恢复,具备较好的纠错能力。

3 结束语

结合公交车辆往复运行的规律,文中提出一种基于单向循环列表的新型公交全自动报站算法。算法采用单向循环列表存储公交往返路线全部站点信息,从而达到了无需判断公交车辆行驶方向的效果,减少了公交司机的工作量;通过周期性的全局搜索进行站点匹配计算,解决了公交车辆突然调头或跨站后出现的自动报站问题,避免了累积误报。该算法经过理论分析和实验验证,具有一定的可行性和有效性。

参考文献:

[1] 杨再龙,赵仁冉,肖 明,等. 基于 GPS 的公交自动报站系统设计[J]. 科技信息,2010(12):236-237.

[2] 彭 可,周 敏,邵 添,等. 一种公交全自动报站方法:CN,201410077708.7[P]. 2014-03-05.

[3] 张国华,黎 明,王静霞. 智能公共交通系统在中国城市的应用及发展趋势[J]. 交通运输系统工程与信息,2007,7(5):24-30.

[4] Ezell S. Explaining international IT application leadership:intelligent transportation systems[C]//Proceedings of IOP conference series:materials science and engineering. [s. l.]:[s. n.],2010.

[5] 李 耀,昂志敏,李敏杰,等. 基于 3G 车载移动终端的 GPS 定位系统设计[J]. 微型机与应用,2012,31(23):51-54.

[6] 王 波. 基于 GPS/BD2 和行驶记录信息的车辆监控终端设计与实现[D]. 杭州:浙江工业大学,2012.

[7] 马丽芳. 基于北斗和 GPRS 车载终端的设计与研究[D]. 西安:西安科技大学,2013.

[8] Senatore S. Special issue on knowledge-intensive fusion for context awareness[J]. Journal of Ambient Intelligence & Humanized Computing,2013,4(4):409-410.

[9] Kawai K,Mcdonald D T. Computer-implemented system and method for identifying duplicate and near duplicate messages:US,US8914331[P]. 2014.

[10] 刘前刚. GPS 定位算法及其在智能公交中的应用[D]. 长沙:湖南大学,2009.

[11] 张 伟. 基于 GPS 和移动互联网的城市公交监管系统设计与研发[D]. 长沙:湖南师范大学,2012.

[12] 彭 勇. 基于 GPS 的公交自动报站算法研究[J]. 通信技术,2009,42(11):211-213.

[13] Razzaque M A,Ahmad S S,Cheraghi S M. Security and privacy in vehicular ad-hoc networks:survey and the road ahead[M]//Wireless networks and security. Berlin:Springer,2013:107-132.

[14] 薛盛可,徐晋鸿,徐晓霞,等. 基于 GPS 的校车自动报站系统设计[J]. 电子技术与软件工程,2014(10):132.

(上接第 147 页)

Or 字符串匹配算法。实验中,通过一台计算机产生数据帧,数据帧通过千兆以太网模块传给 FPGA,在 FPGA 中完成字符串匹配。通过实验验证,结果与理论相符,达到了在千兆以太网下线速实现字符串匹配的目标。

参考文献:

[1] 于 静. 面向 Web 应用的安全服务器网卡的研究与设计[D]. 济南:济南大学,2010.

[2] 郑庆良,张 翔,杨 莹. 网络服务器模型分析与实现[J]. 杭州电子工业学院学报,2004,24(4):95-98.

[3] 黄 建. 入侵检测系统中字符串匹配算法与实现[D]. 武汉:华中科技大学,2008.

[4] Fisk M,Varghese G. An analysis of fast string matching applied to content-based forwarding and intrusion detection[R]. San Diego:University of California,2002.

[5] Knuth D E,Mooris J H,Pratt J V R. Fast pattern matching in strings[J]. SIAM Journal on Comupting,1977,6(2):323-350.

[6] Boyer R S,Moore J S. A fast string searching algorithm[J]. Communications of the ACM,1977,20(10):762-772.

[7] 邱庆哲. 入侵检测系统中检测引擎的研究与实现[D]. 武汉:华中科技大学,2007.

[8] 王 诚,吴继华. Altera FPGA/CPLD 设计[M]. 北京:人民邮电出版社,2005.

[9] Navarr G. 柔性字符串匹配[M]. 中科院计算所网络信息安全研究组,译. 北京:电子工业出版社,2007:15-20.

[10] 孙 鹏,董玉华,韩正之. 基于数据链路层的局域网流量统计的实现[J]. 计算机工程与应用,2002,38(5):150-152.

[11] 王长清,张素娟,蒋景红. 基于以太网帧的嵌入式数据传输方案及实现[J]. 计算机工程与设计,2011,32(6):1952-1956.

[12] Salamon S,Maxmell W M. Storage of ram semen[J]. Animal Reproduction Science,2000,62(1-3):77-111.

[13] Kilts S. 高级 FPGA 设计:结构、实现和优化[M]. 孟宪元,译. 北京:机械工业出版社,2009.

[14] 李桂华,孙仲林,吉利久. CMOS 锁相环 PLL 的设计研究[J]. 半导体杂志,2000,25(3):30-37.