(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 104753934 A
(43)申请公布日 2015.07.01

(21)申请号 201510126647.3

(22)申请日 2015.03.23

(71)申请人 电子科技大学
地址 610041 四川省成都市高新区(西区)西源大道 2006 号

(72)发明人 郝玉洁 周洪川 刘渊 张凤荔 张俊娇

(74)专利代理机构 成都金英专利代理事务所（普通合伙）51218
代理人 袁英

(51)Int.Cl.
H04L 29/06(2006.01)
H04L 29/12(2006.01)

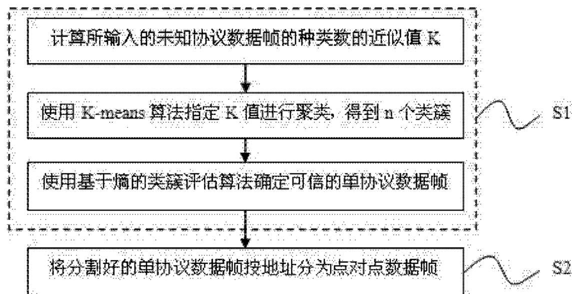
权利要求书3页 说明书14页 附图5页

(54) 发明名称

将未知协议多通信方数据流分离为点对点数据流的方法

(57) 摘要

本发明公开了一种将未知协议多通信方数据流分离为点对点数据流的方法，它包括以下步骤：S1：将混合未知多协议数据流分为单协议数据帧：采用聚类算法将混合未知多协议数据流分为单协议数据帧，并用评估算法确定所得到的类簇是比较可信的单协议数据帧；S2：将分割好的单协议数据帧按地址分为点对点数据帧：通过寻找具有“地址特征”的列队来组成地址对候选集，然后通过对地址对候选集进行拼接，得到最后的地址对。本发明将基于熵值的类簇评估方法用到了协议聚类的评估中，并且提出了一种简单有效的寻找未知协议地址信息的方法，效果很好。



1.将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于：它包括以下步骤：

S1：将混合未知多协议数据流分为单协议数据帧：采用聚类算法将混合未知多协议数据流分为单协议数据帧，并用评估算法确定所得到的类簇是比较可信的单协议数据帧；

S2：将分割好的单协议数据帧按地址分为点对点数据帧：通过寻找具有“地址特征”的列队来组成地址对候选集，然后通过对地址对候选集进行拼接，得到最后的地址对。

2.根据权利要求1所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于：所述的S1包括以下子步骤：

S11：计算所输入的未知协议数据帧的种类数的近似值K，并且得到经过处理的数据帧；

S12：使用K-means算法指定K值进行聚类，得到n个类簇；

S13：使用基于熵的类簇评估算法进行评估每一个类簇的好坏，确定出可信的单协议数据帧。

3.根据权利要求2所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于：所述的S1还包括以下子步骤：

S14：将聚类效果好的类簇放入结果集中，提取该类的指纹信息，并存入指纹库；

S15：将聚类效果好的类簇加上类标识进行机器学习，建立分类模型，使用分类模型进行分类。

4.根据权利要求2所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于：所述的S11包括以下子步骤：

S1101：将输入数据构成二维矩阵，一个字节作为最小处理单元；

S1102：遍历所有字节，计算出每一列中出现频率最高的字符，并分别表示为a1，a2，a3，…，am；同时计算出这些字符在哪些行出现，并分别由集合S1，S2，S3，…Sm表示，即a1为第一列出现频率最高的字符，S1为第一列中出现字符a1的所有行的行号的集合；

S1103：将出现频率大于liminal％的字符以及出现频率小于low_liminal％的字符剔除，设有i个字符符合要求，则对m的值进行更新：令m＝m-i；所述的liminal％和low_liminal％分别为频率最小阈值和频率最大阈值；

S1104：找出集合S1到Sm中，元素个数最大的集合，设为Smax；

S1105：定义一个新的集合R，所述的集合R的元素为集合S，并将集合Smax加入集合R；

S1106：取uniterate的值从50到99，遍历集合S1到Sm，根据遍历到的集合与集合R中所有的集合的交集率，做不同的处理：

(1)若遍历到的集合Sx与集合R中所有的集合的交集率低于uniterate％，则将Sx加入集合R；

(2)若遍历到的集合Sx与集合R中所有的集合的交集率高于或等于uniterate％，则取Smax和Sx的交集作为Smax；

所述的uniterate％为交集率阈值；

S1107：求出R中所有元素的并集，即为筛选出来的数据帧；

S1108：将筛选出来的数据帧从输入数据帧中除去，对剩余的数据帧数量进行判断：

(1) 如果剩余的数据帧数量仍比较大，再次做为输入样本计算这些数据帧的 K 值，即返回步骤 S1101 ；

(2) 否则，进入步骤 S1109 ；

S1109 :得出对应的 K 值，以 uniterate 的值为 X 轴，K 值为 Y 轴作曲线 ；

S1110 :取 K 值变化比较平缓的最大 uniterate 区间，计算在此区间内的 K 的平均值，即为所求的协议种类数的近似值 K。

5. 根据权利要求 2 所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于 :所述的 S12 包括以下子步骤 :

S1201 :将步骤 S11 得到的经过处理的数据帧和簇的近似值 K 输入 ；

S1202 :随机选择 K 个数据对象作为初始聚类中心 ；

S1203 :根据簇中对象的平均值，将每个对象赋给最类似的簇 ；

S1204 :更新簇的平均值，即重新计算每个对象簇中对象的平均值 ；

S1205 :判断聚类准则函数是否收敛即计算聚类准则函数 E 值是否变化 :

(1) 若聚类准则函数未收敛即聚类准则函数 E 值仍在变化，则返回步骤 S1203 ；

(2) 若聚类准则函数收敛即聚类准则函数 E 值没有在变化，则输出 K 个簇。

6. 根据权利要求 2 所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于 :所述的 S13 包括以下子步骤 :

S1301 :将步骤 S11 得到的经过处理的数据帧转换为列二维矩阵，每一个元素为一个字节 ；

遍历所有字节，计算出每一列中的字符的种类表示为 a1, a2, a3, …, ax, 同时计算出 a1 到 ax 中，每个字节出现的概率

S1302 :遍历所有字节，计算出每一列中出现频率最高的字符，并分别表示为 a1, a2, a3, …, am ；同时计算出这些字符在哪些行出现，并分别由集合 S1, S2, S3, …Sm 表示，即 a1 为第一列出现频率最高的字符，S1 为第一列中出现字符 a1 的所有行的行号的集合 ；并将出现的次数除以总行数就得到该字节出现的频率 Pi ；

S1303 :计算每一列的熵值 H，由于有 m 列则有 m 个熵值，计算公式如下 :

$$H = -\sum_{i=1}^{m} P_i lb P_i ;$$

式中，m 为一列中字符的种类数，Pi 为第 i 中字符出现的概率，对数以 2 为底 ；

S1304 :以列号为 X 轴，该列的熵值为 Y 轴做图，分析聚类结果的好坏 :

设定一个评估阈值 low_entropy，当越多的列熵值小于 low_entropy，聚类效果就越好。

7. 根据权利要求 1 所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于 :所述的 S2 包括以下子步骤 :

S21 :将步骤 S1 得到的单协议数据帧输入，并转化为二维数组 ；

S22 :在数据帧中的寻找符合以下条件这些列 :在这些列中，出现字符的种类数大于 1 小于 K，K 作为可变参数，默认值为 256 ；

S23 :循环处理从步骤 S22 中寻找到的每一列，挑选出符合以下条件的列到集合 R :

在其中一个列中，有超过 w% 的字符在另外一个列中的不同位置也出现了，并且在所述的另外一个列中，有超过 w% 的字符在所述的其中一个列中的不同位置也出现了，则将这两列加入集合 R ；所述的 w 作为可变参数，默认值为 60 ；

S24 ：集合 R 中得到的列为地址列的候选集，若集合 R 中不止两列，则将相邻的列进行拼接操作 ；

S25 ：取 w 的值从 10 到 90，分别计算出相应的地址对 ；

S26 ：对比分析得到的地址对，找出最优解。

8. 根据专利要求 2 所述的将未知协议多通信方数据流分离为点对点数据流的方法，其特征在于 ：所述的 S12 采用 weka 工具中的 k-means 聚类算法进行聚类，包括以下子步骤 ：

(1) 数据预处理 ：在将二进制数据流处理成十六进制时，用空格符将每个字节隔开以方便计算，再聚类前使用 weka 自带的 StringToWordVector 工具，将每个字节作为一个属性，一个字节有 256 种形态，因此有 256 个属性 ；过滤所有数据帧，对于每条数据帧，如果出现某个字节，对应的属性值就置为 1，没出现的字节就置为 0，一条数据帧对应一个实例 ；

(2) 使用 weka 实现了的 simplemeans 聚类算法进行聚类，指定 k 值为以上求得的值，聚类出来的结果，就是每种单一的协议类型。

将未知协议多通信方数据流分离为点对点数据流的方法

技术领域

[0001] 本发明涉及一种将未知协议多通信方数据流分离为点对点数据流的方法。

背景技术

[0002] 在当前信息战场景下，被敌方通过进口器件或特种木马进行窃密的威胁日益严峻，此类窃密其途径通常是通过无线通信方式发送涉密信息，且这种通信所采用的协议均为非常规的专用未知协议，而现有的防范措施基本只针对已知协议，大多采用基于端口映射或静态特征匹配等方法，无法对该类窃密渠道进行监测和检测。本课题针对上述问题，拟提出一种基于数据报指纹关系的未知协议发现方法，为该类窃密渠道的监测手段奠定技术基础。

发明内容

[0003] 本发明的目的在于克服现有技术的不足，提供一种将未知协议多通信方数据流分离为点对点数据流的方法，提出了一种简单有效的寻找未知协议地址信息的方法，此方法的前提是得到了单协议数据帧。

[0004] 本发明的目的是通过以下技术方案来实现的：将未知协议多通信方数据流分离为点对点数据流的方法，它包括以下步骤：

[0005] S1：将混合未知多协议数据流分为单协议数据帧：采用聚类算法将混合未知多协议数据流分为单协议数据帧，并用评估算法确定所得到的类簇是比较可信的单协议数据帧；

[0006] S2：将分割好的单协议数据帧按地址分为点对点数据帧：通过寻找具有“地址特征”的列队来组成地址对候选集，然后通过对地址对候选集进行拼接，得到最后的地址对。

[0007] 所述的S1包括以下子步骤：

[0008] S11：计算所输入的未知协议数据帧的种类数的近似值K，并且得到经过处理的数据帧；

[0009] S12：使用K-means算法指定K值进行聚类，得到n个类簇；

[0010] S13：使用基于熵的类簇评估算法进行评估每一个类簇的好坏，确定出可信的单协议数据帧。

[0011] 所述的S1还包括以下子步骤：

[0012] S14：将聚类效果好的类簇放入结果集中，提取该类的指纹信息，并存入指纹库；

[0013] S15：将聚类效果好的类簇加上类标识进行机器学习，建立分类模型，使用分类模型进行分类。

[0014] 所述的S11包括以下子步骤：

[0015] S1101：将输入数据构成二维矩阵，一个字节作为最小处理单元；

[0016] S1102：遍历所有字节，计算出每一列中出现频率最高的字符，并分别表示为a1，a2，a3，…，am；同时计算出这些字符在哪些行出现，并分别由集合S1，S2，S3，…Sm表示，

即 a1 为第一列出现频率最高的字符，S1 为第一列中出现字符 a1 的所有行的行号的集合；

[0017] S1103：将出现频率大于 liminal％的字符以及出现频率小于 low_liminal％的字符剔除，设有 i 个字符符合要求，则对 m 的值进行更新：令 m ＝ m-i；所述的 liminal％和 low_liminal％为频率最小阈值和频率最大阈值；

[0018] S1104：找出集合 S1 到 Sm 中，元素个数最大的集合，设为 Smax；

[0019] S1105：定义一个新的集合 R，所述的集合 R 的元素为集合 S，并将集合 Smax 加入集合 R；

[0020] S1106：取 uniterate 的值从 50 到 99，遍历集合 S1 到 Sm，根据遍历到的集合与集合 R 中所有的集合的交集率，做不同的处理：

[0021] (1) 若遍历到的集合 Sx 与集合 R 中所有的集合的交集率低于 uniterate％，则将 Sx 加入集合 R；

[0022] (2) 若遍历到的集合 Sx 与集合 R 中所有的集合的交集率高于或等于 uniterate％，则取 Smax 和 Sx 的交集作为 Smax；

[0023] 所述的 uniterate％为交集率阈值；

[0024] S1107：求出 R 中所有元素的并集，即为筛选出来的数据帧；

[0025] S1108：将筛选出来的数据帧从输入数据帧中除去，对剩余的数据帧数量进行判断：

[0026] (1) 如果剩余的数据帧数量仍比较大，再次做为输入样本计算这些数据帧的 K 值，即返回步骤 S1101；

[0027] (2) 否则，进入步骤 S1109；

[0028] S1109：得出对应的 K 值，以 uniterate 的值为 X 轴，K 值为 Y 轴作曲线；

[0029] S1110：取 K 值变化比较平缓的最大 uniterate 区间，计算在此区间内的 K 的平均值，即为所求的协议种类数的近似值 K。

[0030] 所述的 S12 包括以下子步骤：

[0031] S1201：将步骤 S11 得到的经过处理的数据帧和簇的近似值 K 输入；

[0032] S1202：随机选择 K 个数据对象作为初始聚类中心；

[0033] S1203：根据簇中对象的平均值，将每个对象赋给最类似的簇；

[0034] S1204：更新簇的平均值，即重新计算每个对象簇中对象的平均值；

[0035] S1205：判断聚类准则函数是否收敛即计算聚类准则函数 E 值是否变化：

[0036] (1) 若聚类准则函数未收敛即聚类准则函数 E 值仍在变化，则返回步骤 S1203；

[0037] (2) 若聚类准则函数收敛即聚类准则函数 E 值没有在变化，则输出 K 个簇。

[0038] 所述的 S13 包括以下子步骤：

[0039] S1301：将步骤 S11 得到的经过处理的数据帧转换为列二维矩阵，每一个元素为一个字节；

[0040] 遍历所有字节，计算出每一列中的字符的种类表示为 a1，a2，a3，…，ax，同时计算出 a1 到 ax 中，每个字节出现的概率

[0041] S1302：遍历所有字节，计算出每一列中出现频率最高的字符，并分别表示为 a1，a2，a3，…，am；同时计算出这些字符在哪些行出现，并分别由集合 S1，S2，S3，…Sm 表示，即 a1 为第一列出现频率最高的字符，S1 为第一列中出现字符 a1 的所有行的行号的集合；

并将出现的次数除以总行数就得到该字节出现的频率Pi；

[0042] S1303：计算每一列的熵值H，由于有m列则有m个熵值，计算公式如下：

[0043] $$H=-\sum_{i=1}^{m}P_i lbP_i$$；

[0044] 式中，m为一列中字符的种类数，Pi为第i中字符出现的概率，对数以2为底；

[0045] S1304：以列号为X轴，该列的熵值为Y轴做图，分析聚类结果的好坏：

[0046] 设定一个评估阈值low_entropy，当越多的列熵值小于low_entropy，聚类效果就越好。

[0047] 所述的S2包括以下子步骤：

[0048] S21：将步骤S1得到的单协议数据帧输入，并转化为二维数组；

[0049] S22：在数据帧中的寻找符合以下条件这些列：在这些列中，出现字符的种类数大于1小于K，K作为可变参数，默认值为256；

[0050] S23：循环处理从步骤S22中寻找到的每一列，挑选出符合以下条件的列到集合R：

[0051] 在其中一个列中，有超过w%的字符在另外一个列中的不同位置也出现了，并且在所述的另外一个列中，有超过w%的字符在所述的其中一个列中的不同位置也出现了，则将这两列加入集合R；所述的w作为可变参数，默认值为60；

[0052] S24：集合R中得到的列为地址列的候选集，若集合R中不止两列，则将相邻的列进行拼接操作；

[0053] S25：取w的值从10到90，分别计算出相应的地址对；

[0054] S26：对比分析得到的地址对，找出最优解。

[0055] 所述的S12采用weka工具中的k-means聚类算法进行聚类，包括以下子步骤：

[0056] (1) 数据预处理：在将二进制数据流处理成十六进制时，用空格符将每个字节隔开以方便计算，再聚类前使用weka自带的StringToWordVector工具，将每个字节作为一个属性，一个字节有256种形态，因此有256个属性；过滤所有数据帧，对于每条数据帧，如果出现某个字节，对应的属性值就置为1，没出现的字节就置为0，一条数据帧对应一个实例；

[0057] (2) 使用weka实现了的simplemeans聚类算法进行聚类，指定k值为以上求得的值，聚类出来的结果，就是每种单一的协议类型。

[0058] 本发明的有益效果是：

[0059] 对于每一步来说，具有以下优点：

[0060] (1) 使用本发明提出的计算混合协议种类数的方法，能够有效的计算出协议种类数的近似值K，这个接下来使用的聚类算法提供很好的参数。

[0061] (2) 从k-means的聚类效果看，使用聚类的方法也能够有效的将不同的协议区分开，k-means算法需要指定不同的随机种子来计算结果的平均值，因为k-means的聚类效果与初始点的选择有很大关系。根据聚类的结果得到的类簇还是比较准的，效果不错。

[0062] (3) 本发明提出的使用熵值来判断一个聚类的类簇的好坏，也有比较好的效果，因为一列的熵值代表了这一列的信息混杂度，如果是同类型的协议帧，按照我们的初始假设，协议存在类型标识，并且类型标识会在同样的位置出现，那么一定存在某列，使得这一列的

熵值很小（接近 0）。

[0063] (4) 本发明提出的寻找未知单协议数据帧中的地址位置原理简单，效果也较好。

[0064] 对于本发明的整体来说，具有以下优点：

[0065] (1) 提出了一种计算混合协议 K 的近似值的方法。

[0066] (2) 将基于熵值的类簇评估方法用到了协议聚类的评估中。由于我们假定协议存在类型标识，并且类型标识会在同样的位置出现，在将输入的二进制流处理为二维矩阵的情况下，使用这种方法对协议帧的评估是非常直观和有效的。

[0067] (3) 提出了一种简单有效的寻找未知协议地址信息的方法，此方法的前提假设是由前面的方法得到了单协议数据帧，效果不错。

附图说明

[0068] 图 1 为本发明流程图；

[0069] 图 2 为实施例 2 中不同 uniterate 时的 K 值变化的示意图；

[0070] 图 3 为实施例 2 中随机种子设为 10 的结果示意图；

[0071] 图 4 为实施例 2 中随机种子设为 5 的结果示意图；

[0072] 图 5 为实施例 2 中随机种子设为 15 的结果示意图；

[0073] 图 6 为 2000 条单协议数据帧每列的熵值图；

[0074] 图 7 为 2500 条多协议混合数据帧每列的熵值图。

具体实施方式

[0075] 下面结合附图进一步详细描述本发明的技术方案：

[0076] 假设：

[0077] (1) 每一种协议都具有协议标识，且同种协议的标识会在相同位置会出现；

[0078] (2) 不同种协议的协议标识可能出现在不同位置，也可能出现在相同位置；

[0079] (3) 协议标识的长度不定（假设不少于 1 字节），可能是 1 字节、2 字节、3 字节…；

[0080] (4) 不同协议的数据帧数量不同，有的多，有的少，甚至有的协议数据帧只有一条。

[0081] 实施例 1 为本发明的具体算法实现：

[0082] 对于步骤 S11，

[0083] 数据输入：n 行 m 列的混合未知协议数据帧。

[0084] 算法目标：尽可能的准确的算出协议的种类数 k。

[0085] 其具体的算法实现：

[0086] (1) 定义最小处理单元对象：OneByte，属性有：

[0087]

```
class OneByte {
    /**此字节所在行*/
    public int row = 0;
    /**此字节所在列*/
    public int line = 0;
    /**此字节在该列中出现的频率*/
    public float frequence = 0f;
    /**此字节的内容*/
    public String oneByte = "";
    /**在该列中，出现此字节的行的编号集合*/
    public HashSet<Integer> alist = new HashSet<Integer>();
    public OneByte(){}
}
```

[0088] (2) 建立 OneByte 的 n 行、m 列的二维数组,将输入的数据帧的每一个字节的内容赋给 OneByte 对象的 oneByte 域,并且记录该字节所在的行和列。

[0089] (3) 循环遍历 OneByte 二维数组,按列统计,统计每一列中每个字节出现的次数以及哪些行出现过该字节。将出现的次数记录下来,记为 num,将出现过的行加入到 OneByte 的 alist 集合中,这样就得到了每个字节在那一列中出现的次数以及出现过该字节的数据帧的行号,出现次数 num 除以 n 就得到该字节出现的频率 frequence。

[0090] (4) 找出每一列中,出现频率最高的 OneByte 对象,从第 0 列到第 m-1 一共有 m 个,同时对这 m 个对象进行筛选,将出现频率 (num/n) 小于 low_liminal 和大于或等于 liminal 的对象去掉,这样就得到了出现频率在 [low_liminal, liminal) 之间的 OneByte 对象,每一个对象都有一个 alist 集合,存放着哪些列出现过该对象。

[0091] (5) 用 S 代表 alist 集合,找出 alist 集合中个数最多的那一个,即为 Smax。

[0092] (6) 用 R 代表结果集存放 S,先将 Smax 加入结果集,用 Si 遍历其他所有的 alist 集合,计算 Si 与 Smax 的交集率 rate,计算公式如下 :

[0093] rate = (Smax 与 Si 交集的个数)/(Smax 中元素个数) ;

[0094] (7) 判断 rate,如果 rate 值小于设定值 uniterate,则将 Si 加入集合 R ;如果 rate 值大于等于设定值 uniterate,则将 Si 与 Smax 求交集,并将新的交集赋给 Smax。

[0095] (8) 结果集 R 中的元素个数,即为要求的一次 k 值。

[0096] 设置 uniterate 值从 50 到 99 变化,分别求出 k 值,以 uniterate 的值为 X 轴,K 值为 Y 轴作曲线。取 K 值变化比较平缓的最大 uniterate 区间,计算在此区间内的 K 的平均值,即为所求的协议种类数的近似值。

[0097] 对于步骤 S12 的具体算法实现 :

[0098] 当由以上方法确定 K 值后,使用 weka 工具中的 k-means 聚类算法进行聚类,操作

流程如下：

[0099]　(1) 数据预处理：在将二进制数据流处理成十六进制时，用空格符将每个字节隔开以方便计算，再聚类前使用 weka 自带的 StringToWordVector 工具，将每个字节作为一个属性，一个字节有 256 种形态，因此有 256 个属性。过滤所以数据帧，对于每条数据帧，如果出现某个字节，对应的属性值就置为 1，没出现的字节就置为 0，一条数据帧对应一个实例；比如，某实例出现了 ff，那么该实例的 ff 属性就设为 1。

[0100]　(2) 使用 weka 实现了的 simplemeans 聚类算法进行聚类，指定 k 值为以上求得的值。聚类出来的结果，就是每种单一的协议类型。

[0101]　对于步骤 S13 的具体算法实现：

[0102]　在使用 K-means 算法对未知协议进行聚类后，对于带有类标签的类别，我们可以知道聚类结果的好坏，但对于完全没有先验知识的类，需要用一种衡量类簇好坏的方法。

[0103]　此算法的计算步骤如下：

[0104]　(1) 将输入数据帧转换为二维矩阵 (n 行，m 列)，每一个元素为一个字节，遍历所有字节，计算出每一列中的字符的种类表示为 a1,a2,a3,…,ax, 同时计算出 a1 到 ax 中，每个字节出现的概率，具体实现按照步骤 S11 中的计算方法的第 1、2、3 步执行。

[0105]　(2) 计算每一列的熵值 H，共有 m 列则有 m 个熵值，计算公式如下：

[0106]
$$H=-\sum_{i=1}^{m}P_i lbP_i\ ;$$

[0107]　其中，m 为一列中字符的种类数，Pi 为第 i 种字符出现的概率，对数以 2 为底。

[0108]　(3) 以列号为 X 轴，该列的熵值为 Y 轴做图，分析聚类结果的好坏。

[0109]　熵值的大小代表了信息混杂程度的大小，在数据帧量很大的情况下，如果是同一种协议的数据帧，那么总有那么一列或几列的熵值接近 0；如果是多种协议混合的，熵值接近 0 的列几乎不会有。因此可以用计算熵值的方法来评估未知协议聚类的好坏，标准就是：设定一个阈值 low_entropy ＝ 0.05，越多的列熵值小于 low_entropy，聚类效果就越好。

[0110]　对于步骤 S2 的具体算法实现：

[0111]　(1) 数据输入：将切分好帧的二进制数据帧转换为对应的十六进制格式，以 2 个字节作为处理单元，构成一个具有 n 行，m 列的二维矩阵，每个元素就是 2 个字节所对应的十六进制字符，用字符串表示。

[0112]　(2) 定义最小处理单元对象：TwoByte，属性有：

[0113]

```
Class  TwoByte {
    /**此字节所在行*/
    public int row = 0;
    /**此字节所在列*/
    public int line = 0;
    /**此字节在该列中出现的频率*/
    public float frequence = 0f;
    /**此字节的内容*/
    public String twoByte = "";
    /**在该列中，出现此字节的行的编号集合*/
    public HashSet<Integer> alist = new HashSet<Integer>();
}
```

[0114] (3) 建立 TwoByte 的 n 行、m 列的二维数组,将输入的数据帧的每二个字节的内容赋给 TwoByte 对象的 twoByte 域,并且记录该字符串所在的行和列。

[0115] (4) 循环遍历 TwoByte 二维数组,按列统计,统计每一列中每个字符串出现的次数以及哪些行出现过该字符串。将出现的次数赋值给 TwoByte 的 num 域,将出现过该字符串的行加入到 TwoByte 的 alist 集合中。这样就得到了每个字符串在那一列中出现的次数以及出现过该字节的数据帧的行号。

[0116] (5) 设定阈值 min_numOfperLine(默认 1) 和 max_numOfperLine(默认 256),筛选出列的字符串种类数大于 min_numOfperLine 且小于 max_numOfperLine 的列作为下一步的输入。

[0117] (6) 假设以上得到 S 列,循环处理每一列,设定阈值 w%(默认 60%)以及结果集 R,挑选出这样的列对到集合 R:

[0118] 在 Si 列中,有超过 w%的字符在 Sj 列中的不同位置也出现了,并且在 Sj 列中,有超过 w%的字符在 Si 列中的不同位置也出现了。则将 Si,Sj 加入集合 R。

[0119] (7) 集合 R 中得到的地址对即为要求的候选地址所在的列。如果集合 R 中不止 2 列,则相邻的列进行拼接。

[0120] (8) 为更准确的找到地址所在的位置,将 w%的值设为从 50 到 95,对比分析 R 中的地址对,找出最优解。

[0121] 实施例 2 为具体的实验验证 :

[0122] 对于步骤 S11,协议种类数计算实验 :

[0123] (1) 数据输入 :Tcpdump 中的 27 种协议,每一种取 100 条数据帧,不够 100 条的全部取 ;每一条数据帧取前 68 字节 ;将所得的协议混合起来作为输入。

[0124] (2) 对可设置变量取值 :liminal,low_liminal,uniterate。Liminal 设为 95,low_liminal 设为 10 ;uniterate 最小值为 50,最大值为 99 ;

[0125] 实验结果：

[0126] 实验 uniterate 取 50 到 99 记录对应的 K 值，以下是 liminal ＝ 95；low_liminal ＝ 10；uniterate ＝ 99 的实验简要结果（一次实验）：

[0127] 帧最大长度为：68；

[0128] 帧总数：2509；

[0129] 列统计器个数：68；

[0130] 候选结果集中集合个数：62；

[0131] 结果集中集合个数：27；

[0132] 字节：00；出现次数：2379；频率：0.9481865；出现的行数：未显示。

[0133] 字节：10；出现次数：1172；频率：0.46711838；出现的行数：未显示。

[0134] 字节：7b；出现次数：700；频率：0.2789956；出现的行数：未显示。

[0135] 字节：38；出现次数：700；频率：0.2789956；出现的行数：未显示。

[0136] 字节：46；出现次数：700；频率：0.2789956；出现的行数：未显示。

[0137] 字节：33；出现次数：700；频率：0.2789956；出现的行数：未显示。

[0138] 字节：10；出现次数：1415；频率：0.56396973；出现的行数：未显示。

[0139] 字节：7b；出现次数：810；频率：0.32283777；出现的行数：未显示。

[0140] 字节：38；出现次数：810；频率：0.32283777；出现的行数：未显示。

[0141] 字节：46；出现次数：810；频率：0.32283777；出现的行数：未显示。

[0142] 字节：33；出现次数：810；频率：0.32283777；出现的行数：未显示。

[0143] 字节：08；出现次数：2279；频率：0.90833；出现的行数：未显示。

[0144] 字节：45；出现次数：2179；频率：0.8684735；出现的行数：未显示。

[0145] 字节：40；出现次数：1368；频率：0.5452371；出现的行数：未显示。

[0146] 字节：80；出现次数：589；频率：0.23475488；出现的行数：未显示。

[0147] 字节：06；出现次数：1340；频率：0.53407735；出现的行数：未显示。

[0148] 字节：ac；出现次数：1635；频率：0.65165406；出现的行数：未显示。

[0149] 字节：10；出现次数：1635；频率：0.65165406；出现的行数：未显示。

[0150] 字节：70；出现次数：995；频率：0.39657235；出现的行数：未显示。

[0151] 字节：64；出现次数：589；频率：0.23475488；出现的行数：未显示。

[0152] 字节：ac；出现次数：1566；频率：0.6241531；出现的行数：未显示。

[0153] 字节：10；出现次数：1566；频率：0.6241531；出现的行数：未显示。

[0154] 字节：70；出现次数：764；频率：0.3045038；出现的行数：未显示。

[0155] 字节：64；出现次数：556；频率：0.22160223；出现的行数：未显示。

[0156] 字节：50；出现次数：1323；频率：0.5273017；出现的行数：未显示。

[0157] 字节：18；出现次数：1143；频率：0.45556；出现的行数：未显示。

[0158] 字节：43；出现次数：275；频率：0.109605424；出现的行数：未显示。

[0159] 结果分析：

[0160] 将 liminal ＝ 95；low_liminal ＝ 10；uniterate 的值设定为从 50 到 99 的结果如下表

[0161]

序号	1	2	3	4	5	6	7	8	9	10	11
uniterate	50	55	60	65	70	75	80	85	90	95	99
K	6	6	12	16	16	19	25	25	25	27	27

[0162] 不同 uniterate 时的 K 值变化的示意图如图 2 所示：

[0163] 根据上表中的数据，取相对最大平滑区间 (80-99) 的平均值：

[0164] K = (25+25+25+27+27)/5 = 25.8

[0165] 因此，k 值取 26。

[0166] 对于步骤 S12，k-means 聚类实验：

[0167] 数据输入：

[0168] Tcpdump 中的 27 种协议，每一种取 100 条数据帧，不够 100 条的全部取；每一条数据帧取前 68 字节；将所得的协议混合起来，每条数据帧后面做上协议类型标记，用于 weka 的 Classes to clusters evaluation 功能评估聚类效果的好坏。

[0169] 操作步骤：

[0170] 1、用 weka 打开 arff 格式文件。

[0171] 2、使用 StringToWordVector 过滤器处理文本属性 StringToVector 的 WordCount 参数设为 flase，其他使用默认的参数设置。处理后数据流的每一个字节表示一个属性，一共有 256 个属性，属性值为 1 或 0，1 表示该属性有，0 表示没有。

[0172] 3、选择 weka 中的 simplemeans 聚类算法进行聚类，选中 Classes to clusters evaluation，算法的 K 值设为 26，随机种子 seed 设为 5，10，15 计算平均值。

[0173] 随机种子设为 10 的结果如图 3 所示，随机种子设为 5 的结果如图 4 所示，随机种子设为 15 的结果如图 5 所示。

[0174] 聚类的总体错误的分类实例数分别为 40.5％，38.1％，33.32％，平均值为 37.2％。也就是有 62.8％的正确率。

[0175] 对于步骤 S13，聚类效果评估实验：

[0176] 设计以下 2 个实验，一个是使用 2000 条单协议数据帧作为输入，另一个是使用 2500 条多协议混合的数据帧作为数据，然后将得到的熵值进行对比分析判断聚类类簇的好坏。

[0177] (1)2000 条单协议每一列的熵值计算如下：

[0178]

列号	熵值	列号	熵值	列号	熵值
1	1.73797	15	0	29	2.923939
2	2.579031	16	0	30	3.635007
3	3.253605	17	0	31	4.842482
4	3.443339	18	0	32	5.652463

5	3.573282	19	0	33	0.677264
6	3.781037	20	0	34	2.003118
7	0.739385	21	0	35	3.112292
8	2.533421	22	1.30097	36	3.222453
9	3.2976	23	0.739385	37	3.317778
10	3.568274	24	2.533421	38	3.374964
11	3.77027	25	3.2976	39	2.923939
12	4.031571	26	3.568274	40	3.635007
13	0	27	3.77027	41	4.83754
14	0	28	4.031571	42	5.654962

[0179] 2000 条单协议数据帧每列的熵值图如图 6 所示。

[0180] (2)2500 条多协议混合数据帧每列的熵值计算如下：

[0181]

列号	熵值	列号	熵值	列号	熵值
1	1.749679	15	0.948731	29	4.242668
2	2.424071	16	1.81334	30	5.41643
3	3.554995	17	1.89479	31	3.047061
4	3.774774	18	7.744579	32	3.279877
5	3.774774	19	8.850714	33	4.820877
6	3.774774	20	8.691174	34	5.511736
7	0.860268	21	1.665774	35	3.738378
8	1.945346	22	0.504124	36	6.786242
9	2.811111	23	3.906183	37	4.319735
10	3.117158	24	2.521798	38	7.148305
11	3.117158	25	9.487966	39	5.022653

12	3.117158	26	9.803289	40	7.954479
13	0.745415	27	2.879997	41	8.894831
14	0.745415	28	3.184749	42	9.403014

[0182] 2500 条多协议混合数据帧每列的熵值图如图 7 所示。

[0183] 从实验结果可以看出：

[0184] (1) 协议混合的熵值最小为 0.504124，最大为 9.803289，而且小于设定阈值 low_entropy 的的列没有；

[0185] (2) 单协议熵最小为 0，最大为 5.654962，而且各个列的值均较小，小于 low_entropy 的列有 9 列；熵值为 0 的，说明该列只有一种字符。

[0186] 由于熵值的大小代表了信息混杂程度的大小，在数据帧量很大的情况下，如果是同一种协议的数据帧，那么总有那么一列或几列的熵值接近 0；如果是多种协议混合的，熵值接近 0 的列几乎不会有。因此可以用计算熵值的方法来评估未知协议聚类的好坏，标准就是：越多的列熵值越小，聚类效果就越好。

[0187] 对于步骤 S2，寻找协议位置信息实验：

[0188] 为验证本算法的有效性，实验分别使用了 2000 条 arp 数据帧和 10000 条 tcp 数据帧分别进行了验证，以下是实验结果。

[0189] (1)2000 条 arp 数据帧地址位置确定实验：

[0190] 数据输入：2000 条 arp 数据帧，取前 42 字节（数据帧最短为 42 字节），2 字节作为最小处理单元，一共有 21 列。

[0191] 实验结果：min_numOfperLine = 1，max_numOfperLine = 256，w%从 50 到 95 的结果如下表（列号从 0 开始）：

[0192]

W%值	候选地址列	地址对	拼接地址对
50	[1, 2, 3, 4, 5, 11, 12, 13,	(1,4)(1,12)(1,17)	[1 2,4 5];[1 2,12 13];[1 2,17 18]

[0193]

	14, 17, 18, 19]	(2,5)(2,13)(2,18) (3,11) (4,1)(4,12)(4,17) (5,2)(5,13)(5,18) (11,3) (12,1)(12,4)(12,17) (13,2)(13,5)(13,18) (14,19) (17,1)(17,4)(17,12) (19,14) (18,2)(18,5)(18,13)	[4 5,12 13]; [4 5,17 18] [12 13,17 18] [17 18 19,12 13 14]
60	[1, 2, 4, 5, 12, 13, 14, 17, 18, 19]	(1,4)(1,12)(1,17) (2,5)(2,13)(2,18) (4,1)(4,12)(4,17) (5,2)(5,13)(5,18) (12,1)(12,4)(12,17) (13,2)(13,5)(13,18) (14,19) (17,1)(17,4)(17,12) (19,14) (18,2)(18,5)(18,13)	[1 2,4 5];[1 2,12 13];[1 2,17 18] [4 5,12 13]; [4 5,17 18] [12 13,17 18] [17 18 19,12 13 14]
70	[1, 2, 4, 5, 12, 13, 14, 17, 18, 19]	(1,4)(1,12)(1,17) (2,5)(2,13)(2,18) (4,1)(4,12)(4,17) (5,2)(5,13)(5,18) (12,1)(12,4)(12,17) (13,2)(13,5)(13,18) (14,19) (17,1)(17,4)(17,12) (19,14) (18,2)(18,5)(18,13)	[1 2,4 5];[1 2,12 13];[1 2,17 18] [4 5,12 13]; [4 5,17 18] [12 13,17 18] [17 18 19,12 13 14]
80	[1, 2, 4, 5, 12, 13, 14, 17, 18, 19]	(1,4)(1,12)(1,17) (2,5)(2,13)(2,18) (4,1)(4,12)(4,17) (5,2)(5,13)(5,18) (12,1)(12,4)(12,17) (13,2)(13,5)(13,18) (14,19) (17,1)(17,4)(17,12) (19,14) (18,2)(18,5)(18,13)	[1 2,4 5];[1 2,12 13];[1 2,17 18] [4 5,12 13]; [4 5,17 18] [12 13,17 18] [17 18 19,12 13 14]
90	[1, 2, 4, 5, 12, 13, 14, 19]	(1,4)(1,12) (2,5)(2,13) (4,1)(4,12) (5,2)(5,13) (12,1)(12,4) (13,2)(13,5) (14,19) (19,14)	[1 2,4 5];[1 2,12 13] [4 5,12 13]
95	[4, 5, 12, 13, 14, 19]	(4,12) (5,13) (12,4) (13,5) (14,19) (19,14)	[4 5,12 13]

[0194]　从上表的拼接地址对可以看出，列号从 0 开始，程序中的 1 2,4 5,12 13,17 18 为地址列。对应于输入数据的列为 :2 3 4 5,8 9 10 11,24 25 26 27,34 35 36 37 为地址列。

[0195]　结果分析 :

[0196]　由上表的结果可以看出，arp 数据帧为地址的列有 :2 3 4 5,8 9 10 11,24 25

26 27,34 35 36 37。

[0197] 分析 arp 数据帧结构,验证实验结果是否正确 :

[0198] 如下是 2 条 arp 数据帧,根据 arp 数据帧的格式可以很容易知道,第 0 1 2 3 4 5 列是目地 MAC 地址,第 6 7 8 9 10 11 列是源 MAC 地址,第 22 23 24 25 26 27 为源 MAC 地址,第 28 29 30 31 为发送方 IP 地址列,第 32 33 34 35 36 37 为目的 MAC 地址列,第 38 39 40 41 为接收方 IP 地址列。

[0199] ff ff ff ff ff ff 00 10 5a 9c b2 54 08 06 00 01 08 00 06 04 00 01 00 10 5a 9c b2 54 ac 10 70 64 00 00 00 00 00 00 ac 10 70 14

[0200] 00 10 5a 9c b2 54 00 c0 4f a3 57 db 08 06 00 01 08 00 06 04 00 02 00 c0 4f a3 57 db ac 10 70 14 00 10 5a 9c b2 54 ac 10 70 64

[0201] 结论 :将算法找出的地址列与输入数据真是的地址列进行比较,虽然没有把所有的地址列都找出来,但是对于每一个地址断,都找出了 2/3 的列,这些列也可以作为将数据帧分离为点对点的依据。

[0202] (2)10000 条 TCP 数据帧地址位置确定实验

[0203] 数据输入 :10000 条 TCP 数据帧,取前 60 字节 (数据帧最短为 60 字节),2 字节作为最小处理单元,一共有 30 列。

[0204] 实验结果 :min_numOfperLine = 1, max_numOfperLine = 256, w%从 50 到 95 的结果如下表 :

[0205]

W%	候选地址列	地址对	拼接地址对
50	[0, 1, 2, 3, 4, 5, 13, 14, 15, 16]	(0,3) (1,4) (2,5) (3,0) (4,1) (5,2) (13,15) (14,16) (15,13) (16,14)	[0 1 2,3 4 5] [13 14,15 16]
60	[0, 1, 2, 3, 4, 5, 13, 14, 15, 16]	(0,3) (1,4) (2,5) (3,0) (4,1) (5,2) (13,15) (14,16) (15,13) (16,14)	[0 1 2,3 4 5] [13 14,15 16]
70	[0, 1, 2, 3, 4, 5, 13, 14, 15, 16]	(0,3) (1,4) (2,5)	[0 1 2,3 4 5] [13 14,15 16]

[0206]

		(3,0) (4,1) (5,2) (13,15) (14,16) (15,13) (16,14)	
80	[1, 2, 4, 5, 13, 14, 15, 16]	(1,4) (2,5) (4,1) (5,2) (13,15) (14,16) (15,13) (16,14)	[1 2,4 5] [13 14,15 16]
90	[13, 14, 15, 16]	(13,15) (14,16) (15,13) (16,14)	[13 14, 15 16]
95	[13, 14, 15, 16]	(13,15) (14,16) (15,13) (16,14)	[13 14, 15 16]

[0207] 从上表的拼接地址对可以看出,程序中的 0 1 2,3 4 5 ,13 14,15 16 为地址列。对应于输入数据的列为 :0 1 2 3 4 5,6 7 8 9 10 11,26 27 28 29,30 31 32 33 为地址列。

[0208] 如下是 2 条 tcp 数据帧,根据 tcp 数据帧的格式可以很容易知道,第 0 1 2 3 4 5 列是目地 MAC 地址,第 6 7 8 9 10 11 列是源 MAC 地址,第 26 27 28 29 为发送方 IP 地址列,第 38 39 40 41 为接收方 IP 地址列。

[0209] 00 10 7b 38 46 33 00 10 5a 9c b2 54 08 00 45 00 00 2c 7c 00 40 00 80 06 81 24 ac 10 70 64 ce fb 12 37 04 18 00 50 00 05 00 94 00 00 00 00 60 02 20 00 75 7e 00 00 02 04 05 b4 05 b4

[0210] 00 10 5a 9c b2 54 00 10 7b 38 46 33 08 00 45 00 00 2c 4b 0a 00 00 3f 06 33 1b ce fb 12 37 ac 10 70 64 00 50 04 18 46 74 b0 bf 00 05 00 95 60 12 7f e0 1e 59 00 00 02 04 05 b4 00 00

[0211] 结论 :将算法找出的地址列与输入数据真是的地址列进行比较,算法所找出的列正好全部是 tcp 数据帧的地址列,这些列可以作为将数据帧分离为点对点的依据。

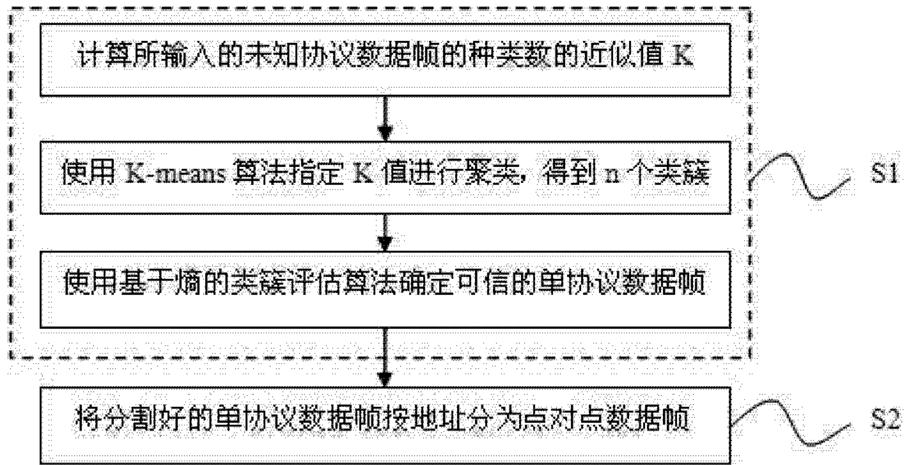


图 1

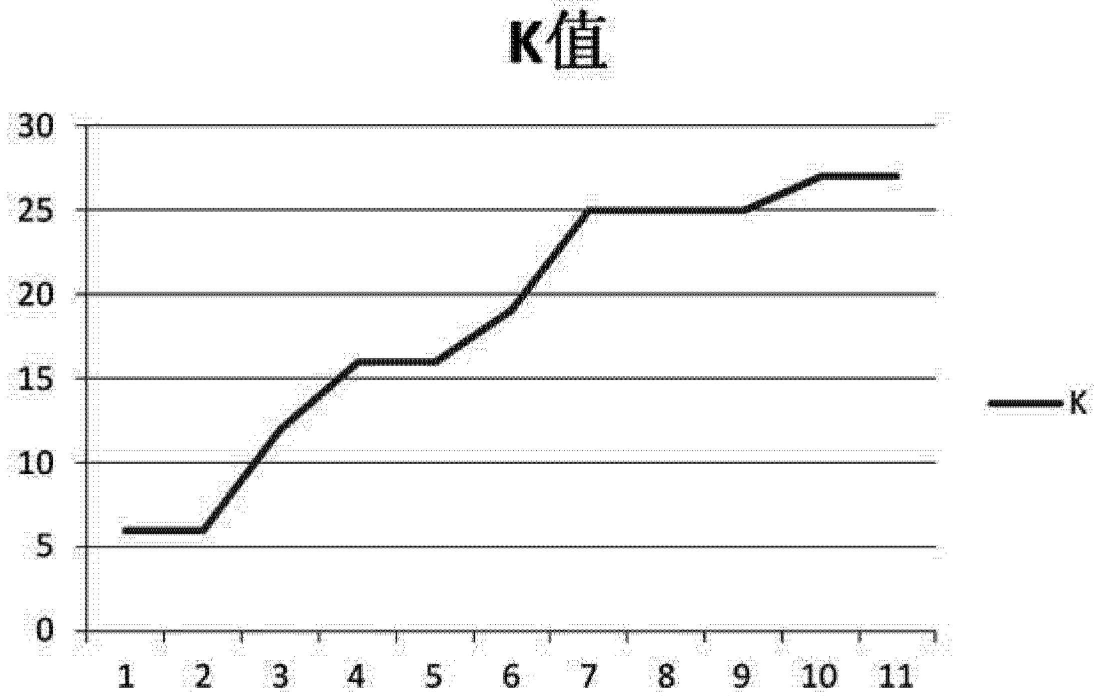


图 2

```
Cluster  0 <-- http
Cluster  1 <-- lpd_or_data
Cluster  2 <-- No class
Cluster  3 <-- ftp-data
Cluster  4 <-- nbss_smb
Cluster  5 <-- loop_data
Cluster  6 <-- nbdgm_smb_browser
Cluster  7 <-- ssh
Cluster  8 <-- llc_cdp
Cluster  9 <-- arp
Cluster 10 <-- telnet
Cluster 11 <-- irc
Cluster 12 <-- ftp
Cluster 13 <-- http_image
Cluster 14 <-- smtp
Cluster 15 <-- No class
Cluster 16 <-- nbss_smb_dcerpc
Cluster 17 <-- pop
Cluster 18 <-- dns
Cluster 19 <-- nbns
Cluster 20 <-- No class
Cluster 21 <-- smb_netlogon
Cluster 22 <-- icmp_errors
Cluster 23 <-- No class
Cluster 24 <-- tcp
Cluster 25 <-- nbss

Incorrectly clustered instances :       1017.0    40.5341 %
```

图 3

```
Cluster  0 <-- No class
Cluster  1 <-- ftp-data
Cluster  2 <-- No class
Cluster  3 <-- lpd_or_data
Cluster  4 <-- nbns
Cluster  5 <-- irc
Cluster  6 <-- nbss_smb
Cluster  7 <-- icmp_errors
Cluster  8 <-- ntp
Cluster  9 <-- tcp
Cluster 10 <-- http
Cluster 11 <-- No class
Cluster 12 <-- telnet
Cluster 13 <-- nbss
Cluster 14 <-- nbss_smb_dcerpc
Cluster 15 <-- No class
Cluster 16 <-- llc_cdp
Cluster 17 <-- loop_data
Cluster 18 <-- ftp
Cluster 19 <-- nbdgm_smb_browser
Cluster 20 <-- smtp
Cluster 21 <-- icmp_data
Cluster 22 <-- pop
Cluster 23 <-- http_image
Cluster 24 <-- ssh
Cluster 25 <-- No class

Incorrectly clustered instances :       957.0    38.1427 %
```

图 4

```
Cluster  0 <-- smb_netlogon
Cluster  1 <-- No class
Cluster  2 <-- time
Cluster  3 <-- ntp
Cluster  4 <-- nbns
Cluster  5 <-- telnet
Cluster  6 <-- irc
Cluster  7 <-- dns
Cluster  8 <-- ftp
Cluster  9 <-- icmp_data
Cluster 10 <-- ftp-data
Cluster 11 <-- nbdgm_smb_browser
Cluster 12 <-- nbss_smb
Cluster 13 <-- http
Cluster 14 <-- arp
Cluster 15 <-- pop
Cluster 16 <-- ssh
Cluster 17 <-- llc_cdp
Cluster 18 <-- syslog
Cluster 19 <-- nbss_smb_dcerpc
Cluster 20 <-- smtp
Cluster 21 <-- icmp_errors
Cluster 22 <-- lpd_or_data
Cluster 23 <-- http_image
Cluster 24 <-- No class
Cluster 25 <-- No class

Incorrectly clustered instances :      836.0    33.32   %
```

图 5

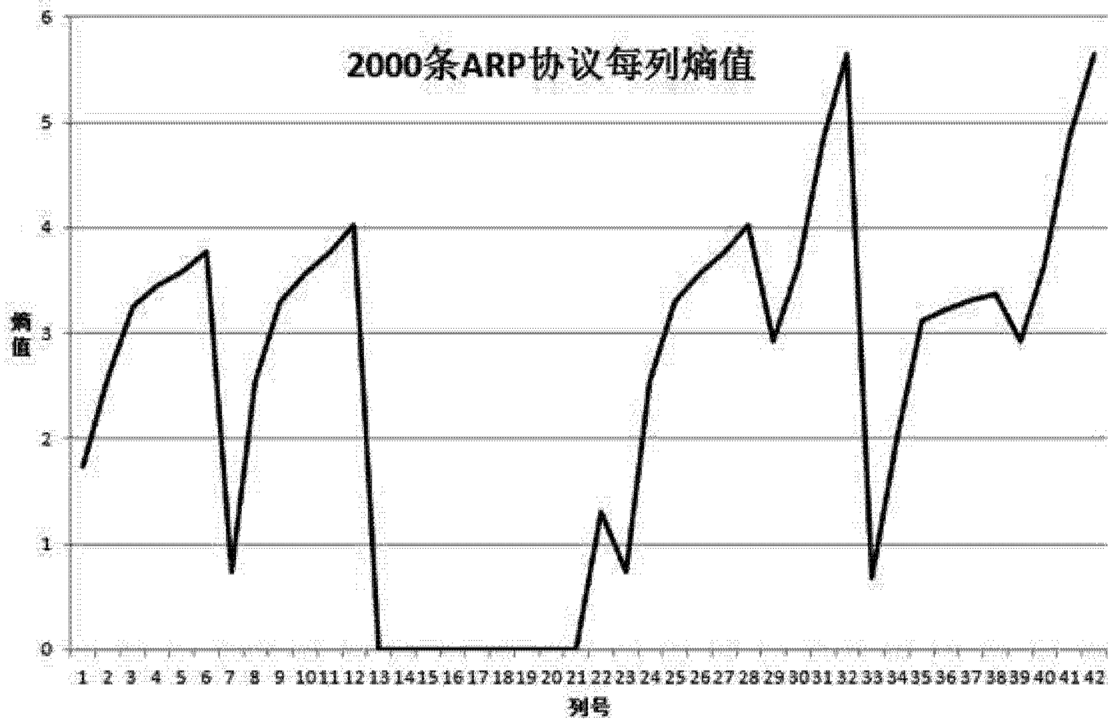


图 6

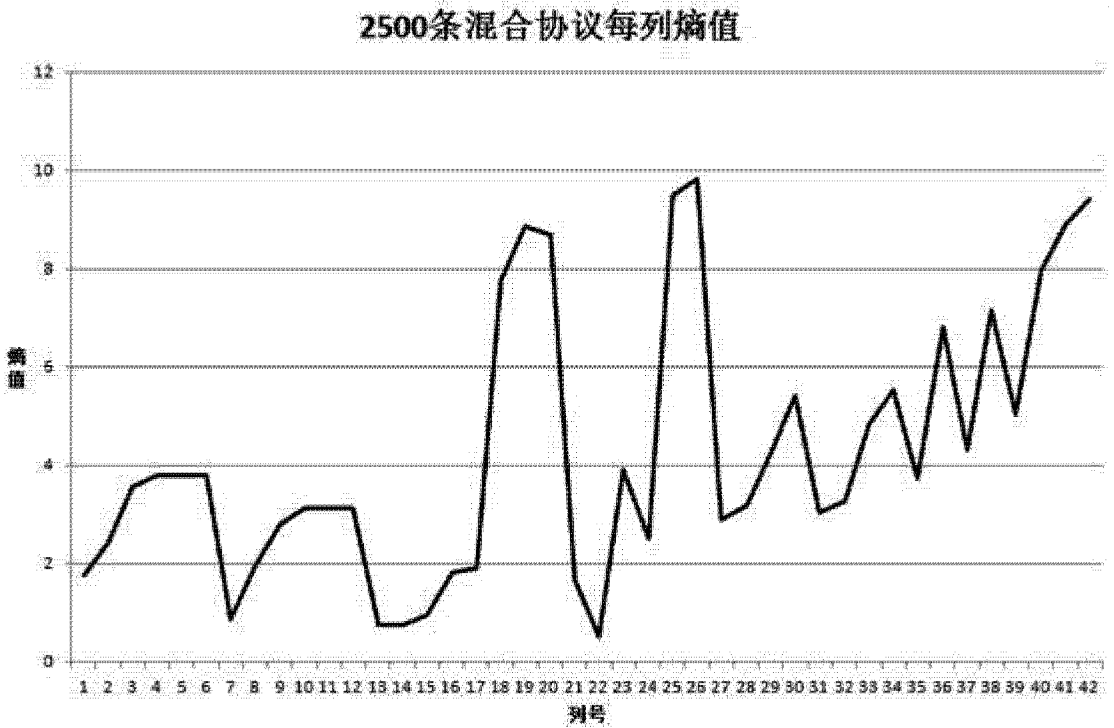


图 7