一种高效的最短路径树动态更新算法

刘代波[1] 侯孟书[1] 武泽旭[2] 屈 鸿[1]
(电子科技大学计算机科学与工程学院 成都 610000)[1] (四川大学电气信息学院 成都 610000)[2]

摘 要 计算动态环境下最短路径树是一个典型的组合优化问题。Ball-and-String 模型是一种高效的动态更新算法,但仍存在不少冗余计算。针对 Ball-and-String 算法中边的处理进行了优化,从而提高了动态更新的效率,同时实现了对节点的删除和增加,以适应最短路径树的拓扑变化。实验结果表明新算法效率更高。

关键词 动态计算,最短路径树,路由,算法
中图法分类号 TP393 **献标识码** A

Efficient Dynamic Algorithm for Computation of Shortest Path Tree

LIU Dai-bo[1] HOU Meng-shu[1] WU Ze-xu[2] QU Hong[1]
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610000, China)[1]
(School of Electrical Engineering and Information, Sichuan University, Chengdu 610000, China)[2]

Abstract The computation of shortest path tree in dynamic environment is a typical combinatorial optimization problem. Ball-and-String Model is an efficient algorithm which can dynamically update shortest path tree(SPT), but exists redundant computation. This paper presented an a new dynamic SPT algorithm that optimizes the processing of edges in Ball-and-String Model. New algorithm raises the efficiency of dynamically updating SPT. Additionally, new algorithm implements deleting of node or adding of node in SPT, accordingly, can adjust for the topological variation of SPT. Experimental results show that new algorithm is more efficient than Ball-and-String Model.

Keywords Dynamic computing, Shortest path tree, Routing, Algorithm

1 引言

互联网中,路由器通过查询路由表来实现数据报的投递。为了减小网络中数据交换的开销,需要找到路由节点之间的最短路径。E. Dijkstra[1] 和 R. Bellman[2] 提出了两种经典的最短路径算法,但是在网络拓扑发生较小变化时,用这两种算法重构最短路径树会浪费大量的路由器时间。而用动态算法更新最短路径树,就可以减小路由器不必要的开销。此法在当前最短路径树的基础上对路由表中潜在的需要更新的部分进行更新,对不受影响的不予处理。动态更新最短路径树算法较静态算法收敛更快、占用路由器计算资源更少。本文在 Ball-and-String 模型[10] 的基础上,提出一种更高效的新算法。

2 相关工作

由于静态 Dijkstra 算法不能有效地适应网络拓扑的变化,因此在其基础上提出了一系列动态更新算法。

文献[3-6]提出的动态算法由静态 Dijkstra 算法转换而来。其中文献[3]最早提出动态更新最短路径树(DSPT)算法,但此算法低效、复杂。文献[4]提出的 DSPT 算法较静态 Dijkstra 算法高效,但是缺乏算法分析和实验数据。文献[7-13]在文献[1]的基础上考虑了文献[3-6]的不足,提出了在当前 SPT 的基础上重构 SPT 的算法。其中文献[8,9]提出的算法只适用于单边的变化,文献[11]提出的算法是针对多边权值减少的情况,文献[13]对文献[10]提出的算法进行了优化。

文献[10]提出的基于 Ball-and-String 模型的算法(下文用 Ball-and-String 表示)是高效的动态更新算法,但是在对边的处理上存在明显的冗余,且与上述算法一样,没有实现对增加节点和删除节点的处理。本文在文献[10]的基础上提出了效率更高的算法,实现了对删除节点和增加节点的处理。

3 新算法

3.1 符号和定义

为了更好地理解该算法,本节对涉及到的符号进行定义和说明。

$G=(V,E)$表示有向图,V 为图中节点的集合,E 为边的集合。设 e 为图中的边 $i\rightarrow j$,$W(e)$为 e 的权值,也可表示为 $W\langle i,j\rangle$。$E(e)$是 e 的头节点,$S(e)$是 e 的尾节点。设 v 为图中的节点,N 是节点集合,$D(v)$表示节点 v 到根节点的最短路径值,$T(v)$表示在最短路径树中以 v 为根节点的子树上的节点集合。$In(v)$是节点 v 的入边集合,$Out(v)$是节点 v 的出边集合。$In(N)$为节点集 N 的入边集合,$In(N)=\{e|E(e)\in$

到稿日期:2010-08-16 返修日期:2010-11-11 本文受国家自然科学基金(60905037),电子科技大学青年基金(L08010601)资助。
刘代波(1985—),男,硕士生,主要主要研究方向为计算机网络、人工智能,E-mail:liudaibo1985@163.com;侯孟书(1971—),男,博士后,副教授,主要研究方向为计算机网络、分布式系统、无线传感网络。

$N\}$，$Out(N)$为节点集 N 的出边集合，$Out(N)=\{e|S(e)\in N\}$。$P(v)$表示节点 v 在最短路径树中的父节点。

MIN$\{N\}$表示数值集合 N 中的最小元素。Q 是一个优先队列，ENQUEUE(Q,$\langle n,p,x\rangle$)表示入队列，p 表示节点 n 的候选父节点，x 表示 p 作为 n 的父节点时节点 n 的最短路径值的增量。exactMin(Q)表示出队列。getItem(Q,v)从队列中提取节点为 v 的项。

3.2 算法描述

边的权值发生变化时，首先确定 SPT 受影响的区域，然后更新区域中的节点。如果受影响区域是增长的(边权值减小这种情况)，则保证每次处理的节点位于区域内。如果 SPT 上的节点 v 的最短路径值发生了改变，那么 $T(v)$中节点的最短路径值都加上相同的增量。更新时以 SPT 上的分支为单位，一次更新整个分支。所有待更新节点以最短路径值增量的大小为序来处理，增量越小优先级越高。

SPT 上的节点都用 black 或 white 来标志其状态。节点的状态信息在更新最短路径树时用来区分待更新节点是可以更新还是无需更新的，状态 white 表示节点可以更新，状态 black 表示节点已经固定，无需更新。

本文从以下 3 方面对 Ball-and-String 算法进行优化。首先，在边 e 的权值增量为正时，如果 $T(E(e))$的入边使得状态为 white 的节点的最短路径值的增量小于 0，则入队列。其次，如果 $T(n)$为被更新的一个分支，则 $\forall e\in \mathrm{Out}(T(n))$。如果 $E(e)$ 已经存在于优先队列中，则选择增量值小的存在于队列中。最后，本文还实现了对增加和删除节点的处理。具体操作见算法流程。

算法 1

Step1 initialize SPT from $G=(V,E)$

Step2 检测 SPT 拓扑变化

1. 如果边的权值发生变化，go to Step3。

2. 如果删除或增加节点，go to Step4。

Step3 e 的权值改变

$u=E(e)$，$v=S(e)$，$inc=D(v)+W(e)-D(u)$，$N=\varphi$

If($inc>0$) //权值增加

If($v!=P(u)$) return;

初始化 SPT 上节点为 black。对 $T(u)$上的节点，修改最短路径值，修改状态为 white 并归入节点集 N。

ENQUEUE(Q,$\langle u,v,inc\rangle$);

$\forall e\in \mathrm{In}(N)$ &&($(D(S(e))+W(e))-D(E(e)))<0$

ENQUEUE(Q,$\langle E(e),S(e),(D(S(e))+W(e)-D(E(e)))\rangle$);

Go to Step5;

If($inc<0$) //权值减小

If($v!=P(u)$&&$D(u)==\min\{D(b)+W\langle b,u\rangle|\langle b,u\rangle\in \mathrm{In}(u)\}$) return;

初始化 SPT 上节点为 white。对 $T(u)$上的节点修改最短路径值，修改状态为 black 并归入节点集 N。

$\forall e\in Out(N)$ &&($(D(S(e))+W(e)-D(E(e)))<0$;

ENQUEUE(Q,$\langle E(e),S(e),((D(S(e))+W(e))-D(E(e)))\rangle$)

Go to Step5;

Step4 如果新增加节点 u:

生成 u 的出边和入边，同时确定 u 的父节点，计算 u 的最短路径值。

$\forall v\in\{E(e)|e\in \mathrm{Out}(u)\}$ //处理出边连接的节点

If $D(v)>D(u)+W\langle u,v\rangle$

Inc=$D(u)+W\langle u,v\rangle-D(v)$;

Go to step3;

如果节点 u 被删除：

把 u 的入边和出边设置为不可达

$\forall v\in\{E(e)|e\in Out(u)\}$ && $P(v)==u$ //处理 u 的子节点

$Inc=\min\{D(b)+W\langle b,v\rangle|\langle b,v\rangle\in In(v)\&\&b!=u\}-D(v)$;

Goto Step3;

Step5 While($Q!=\varphi$)

$\langle n,p,x\rangle\leftarrow$exactMin($Q$);$P(n)=p$;

更新 $T(n)$中任意节点的最短路径值，修改状态为 black；如果 $T(n)$中的节点存在于队列 Q 中，则从队列中删除含此节点的项。

$\forall e\in \mathrm{Out}(T(n))$ && $state(E(e))==$white; //处理出边

If($(D(S(e))+W(e)-D(E(e)))<0$ && $E(e)$ is not in Q)

ENQUEUE(Q,$\langle E(e),S(e),((D(S(e))+W(e))-D(E(e)))\rangle$);

Else if($(D(S(e))+W(e)-D(E(e)))<0$)

$\langle n,p,x\rangle\leftarrow$getItem($Q$,$E(e)$);

如果 $D(S(e))+W(e)-D(E(e)))<x$，更新队列中的项。

return;

3.3 实例测试

本节用实例来说明新算法的运行流程。图 1 中边$\langle b,g\rangle$的权值由 3 增加到 10，增量为 7。以 g 为根的子树上的节点 $g,i,j,k,l,n,o,p,q,r,t,u,v,w$ 受到影响。更新这些节点的最短路径值和状态。在这些节点的入边中，把边$\langle b,g\rangle$和使得尾节点的最短路径值增量小于 0 的边入队列，如表 1 所列。

表 1 可能改变 SPT 的边

边	⟨f,l⟩	⟨m,o⟩	⟨m,n⟩	⟨c,g⟩	⟨h,i⟩
节点	l	o	n	g	i
增量	−3	−2	−5	−1	−

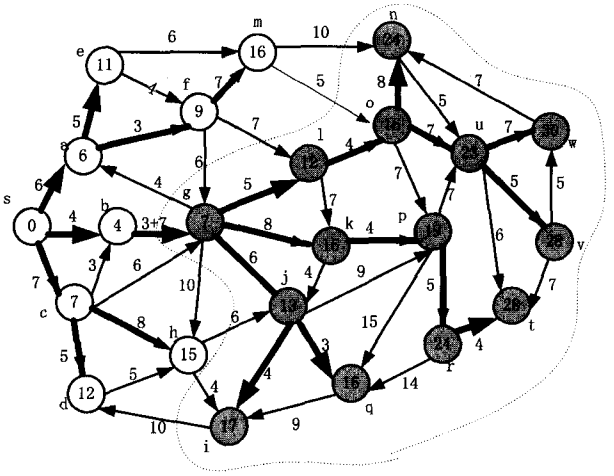

图 1 边$\langle b,g\rangle$发生变化之前的 SPT

边$\langle m,n\rangle$首先出队列，对以 n 为根节点的子树上的节点(只有 n 一个节点)的最短路径值加上增量−5 并且修改 n 的

父节点为 m。节点 n 的出边$\langle n,u\rangle$使得节点 u 的最短路径值增量为 1，不需要处理。边$\langle h,i\rangle$的处理与边$\langle m,n\rangle$的处理一样。然后处理边$\langle f,l\rangle$，以 l 为根的子树上的节点 l,o,u,v,w 的最短路径值都加上增量−3，并且修改节点 l 的父节点为 h。出边$\langle u,t\rangle$使得 t 的最短路径值增量为−2，入队列。当队列为空，本次更新结束。更新之后的 SPT 见图 2，深色节点为被更新过的节点。

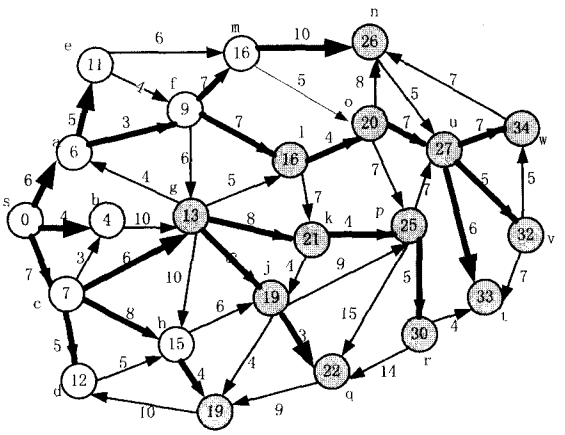

图 2　更新之后的 SPT

4　复杂度分析

新算法分为边的权值的增加和减少、节点的增加和删除 4 个处理模块。各个模块的时间复杂度比较接近。本文选择边权值增加的情况来分析算法的时间复杂度。设边 e(即$\langle u,v\rangle$)权值的增加使得当前 SPT 需要更新，假设存在一个因子 λ 使得 $T(v)$的节点数不大于 λN(N 为 SPT 节点数)，设存在因子 κ 使得节点集合的入边数(经过筛选的有可能会影响节点集的最短路径的边)不大于节点集的节点总数乘以因子 κ，同时假设存在常数 α 使出边数不大于节点集中节点总数乘以 α。又设节点集中每个子分支上的节点数不大于节点集中节点总数乘以常数因子 μ。所有的更新都经过优先队列。由于队列在增长的同时也不断地出队列，因此队列的最大长度不会超过 $T(v)$的节点总数。优先队列用堆来实现，队列操作最坏的时间复杂度由 $O(T(v))$减小到 $O(\log T(v))$。根据上面的分析，边权值增加时更新最短路径树的总的时间复杂度为 $O(\lambda N)+O(\log(\kappa\lambda N)(\mu\lambda N+\alpha\lambda N))$，$\mu,\lambda,\kappa,\alpha$ 是介于 0 与 1 之间的常数。最坏的情况是所有因子都为 1，所以总的时间复杂度不会超过 $O(N\log(N))$。

5　仿真实验

本节用 3 个实验展示新算法的性能。实验 1 分别用静态 Dijkstra 算法和动态更新算法初始化 SPT。实验 2 比较在边的权值发生变化时 Ball-and-String 算法和新算法的动态更新效率。实验 3 展示新算法在增加节点和删除节点时的更新效率。

PC 配置为 Intel(R) Pentium(R) 4 CPU 2.66GHz，内存 512MB。运行在 RedHat enterprise5 上，使用 GNU C++编程实现。

实验 1 在连通图上初始化 SPT。静态 Dijkstra 算法和动态算法(新算法与 Ball-and-String 算法的初始化相同)在图的节点数分别为 100、1000、1 万、10 万、20 万和 30 万时构造 SPT 的时间开销见表 2。实验表明，使用静态 Dijkstra 算法初始化 SPT，效率更高。

表 2　各算法初始化 SPT 所需时间

节点数	Dijkstra 算法	动态算法
100	0.5ms	0.9ms
1000	4.6ms	14ms
1 万	93ms	208ms
10 万	1590ms	3800ms
20 万	2625ms	8899ms
30 万	4386ms	>10000ms

但是，当 SPT 上边的权值发生变化或者节点发生增减时，也用静态 Dijkstra 算法更新 SPT，代价将非常大。对于节点数为 1 万的 SPT，更新 100 次将花费约 10s 的时间。实际上，随着节点的增加，SPT 拓扑发生变化的频率会增大。所以，当 SPT 的拓扑发生变化时，不使用静态 Dijkstra 算法来更新。本文使用 Ball-and-String 与新动态算法来更新 SPT 并进行效率的比较。

实验 2 分为 4 部分，分别在节点规模为 100～1000、1000～1 万、1 万～10 万和 10 万～30 万时使用 Ball-and-String 和新算法对 SPT 动态更新。由于每一次更新的时间非常短，因此实验中给出的数据是 1 万次更新时间的总和，时间单位为 ms。

首先 SPT 的节点规模在 100～1000 之间时，使用 Ball-and-String 算法与新算法动态更新 SPT。时间开销见图 3，其中红线为新算法花费的时间，黑线为 Ball-and-String 算法花费的时间。由实验结果可以看出，Ball-and-String 更新 1 万次用时在 35ms 至 50ms 之间，而新算法在 15ms 至 40ms 之间。

其次，当 SPT 的节点规模在 1000～1 万之间时，使用 Ball-and-String 与新算法动态更新 SPT 的时间开销见图 4。

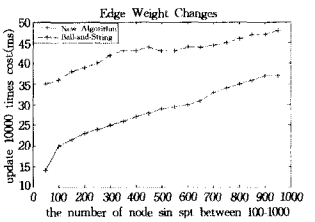

图 3　SPT 节点规模在 100～1000 之间的更新时间

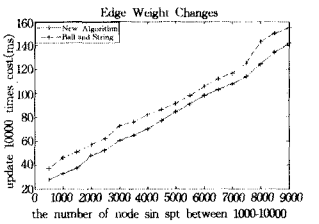

图 4　SPT 节点规模在 1000～1 万之间的更新时间

再次，SPT 的节点规模在 1 万～10 万之间时，使用 Ball-and-String 与新算法动态更新 SPT 的时间开销见图 5。

最后，当 SPT 的节规模达到数十万时，更新 SPT 的时间也急剧增加。鉴于实验设备的局限性，我们只选择节点规模为 10 万、20 万和 30 万的 SPT 进行更新。实验分析见图 6。

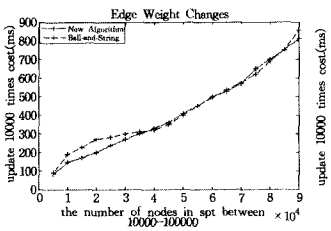

图 5　SPT 节点规模在 1 万～10 万之间的更新时间

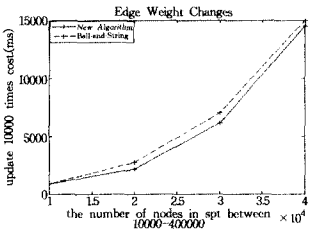

图 6　SPT 节点规模在 10 万～30 万之间的更新时间

从以上实验数据可以看出，新算法效率更高。在此基础

上，新算法还实现了在 SPT 中增加和删除节点。需要表明的是，删除节点的操作只是修改图中与此节点连接的边的权值为无穷大，而增加节点则需要在图中生成新边，所以增加节点较删除节点消耗更多的时间。

实验 3 在 SPT 节点规模为 1 万～10 万之间进行。实验分两部分，第一部分在 SPT 上连续删除 1000 个节点；第二部分在 SPT 上增加 100 个节点。第一部分实验数据见图 7，时间单位为 μs。第二部分实验数据见图 8，时间单位为 ms。

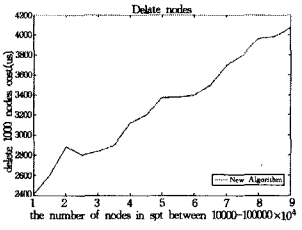

图 7 新算法在 SPT 上删除节点

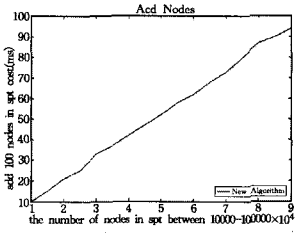

图 8 新算法在 SPT 上增加节点

结束语 新算法在 Ball-and-String 的基础上做了进一步优化。由实验数据可以看出，新算法在边的权值发生变化时更新 SPT 的效率更高。在功能上，新算法还实现了对节点的增加和删除，适合处理 SPT 的任何变化。

参 考 文 献

[1] Dijkstra E. A note two problems in connection with graphs[J]. Numerical Mathemat.,1959,1:269-271

[2] ElBlman R. On a routing problem [J]. Quarterly Appl. Mathemat.,1958,16:87-90

[3] Spira P, Pan A. On finding and updating spanning trees and shortest paths[J]. SIAM J. Computing,1975,4(3):375-380

[4] McQuillan J, Richer I, Rosen E. The new routing algorithm for the ARPANET[J]. IEEE Trans. Commun,1980,COM-28:711-719

[5] Franciosa P, Frigioni D, Giaccio R. Semi-dynamic shortest paths and breadth-first search in digraph [C] // Proc. 14th Annu. Symp. Theoretical Aspects of Computer Science. Mar. 1997:33-46

[6] Barbenn M, Hutchinson S. Increment algorithms for single-source shortest path trees[C] // Process Foundations of Software Technology and Theoretical Computer Science. 1994:113-124

[7] Xiao B, Cao Jiang-nong, Shao Z, et al. An Efficient Algorithm for Dynamic Shortest PathTree Updatein Network Routing[J]. Journal of Communications and Networks,2007,9(4)

[8] Narvaez P, Siu K-Y, Tzeng H-Y. New dynamic algorithms for shortest path tree computation[J]. IEEE Trans. Networking, 2000,8

[9] Xiao B, Zhuge Q, Sha E H-M. Minimum dynamic update for shortest path tree construction[C] // Proceedings of the GLOBECOM'01. Nov. 2001:126-130

[10] Narvaez P, Siu K-Y, Tzeng H-Y. New dynamic SPT algorithm based on a ball-and-string model[J]. IEEE/ACM Trans. Networking,2001,9:706-718

[11] Xiao B, Cao Jian-nong. Dynamic shortest path tree update for multiple link state decrements[C] // 2004 Global Telecommunications Conference. 2004

[12] Xiao B, Cao J, Zhuge Q, et al. Dynamic update of shortest path tree in OSPF[C] // Parallel Architectures, Algorithms and Networks. May 2004:18-23

[13] Chan E P F, Yang Ya-Ya. Shortest Path Tree Computation in Dynamic Graphs[J]. IEEE Transactions on Computers,2009,58(4):541-557

(上接第 95 页)

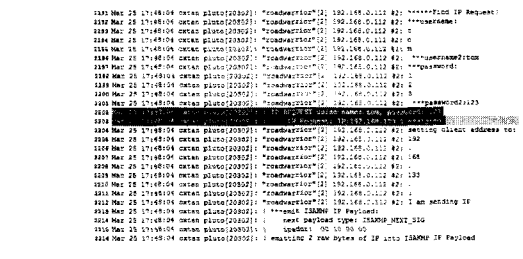

图 7 IP 地址分配程序执行过程截图

结束语 本文针对目前基于标准 IKE 的移动终端远程安全接入方案的缺陷，通过在 IKE 第 1 阶段主模式的第 5 条消息中增加用户身份载荷，接入网关根据用户身份分配合适的内网 IP 添加在第 6 条消息中，实现了对用户远程安全接入获得内网信息的支持，而且扩展了 IKE 的认证方式。根据身份分配内网 IP 的方式更有利于访问控制管理。而且，由于扩展载荷增加在第 5，6 条消息中，属于加密传输，不会降低 IKE 协商的安全性。因此，本方案不仅满足远程客户的安全接入需求，而且具有更高的协商效率、更强的可控性及灵活性，可以有效地满足用户对移动办公的安全需求。

参 考 文 献

[1] Davis C R. IPSec VPN 的安全实施[M]. 周永彬，冯登国，徐震，等译. 北京：清华大学出版社，2002

[2] Kent S, Seo K. Security Architecture for the Internet Protocol [S]. RFC 4301, Dec. 2005

[3] Harkins D, Carrel D. The Internet Key Exchange (IKE) [S]. RFC 2409, Nov. 1998

[4] Zhao A-qun, Yuan Yuan, Ji Yi, et al. Research on tunneling techniques in virtual private networks[C] // Proceedings of Communication Technology(WCC-ICCT 2000). 2000:691-697

[5] Hue C A, Kalafut A J, Gupta M. A Unified Approach to Intra-domain Security[C] // Computational Science and Engineering (CSE '09). 2009:219-224

[6] Maughhan D, Sehertler M, Sclmeider M, et al. Internet Security Association and Key Management Protocol (ISAKMP) [S]. RFC2408, Nov. 1998

[7] Orman H. The OAKLEY Key Determination Protocol [S]. RFC2412, Nov. 1998

[8] Krawczyk H. SKEME: A Versatile Secure Key Exchange Mechanism for Internet[C] // IEEE Proceedings of the 1996 Symposium on Network and Distributed Systems Security. 1996

[9] Kaufman C. Internet Key Exchange(IKEv2) Protocol[S]. IETF RFC4306, Sep. 2004

[10] Wouters P, Bantoft K. Building and Integrating Virtual Private Networks with Openswan[EB/OL]. http://www.openswan.org/,2009

[11] Ren Qing-liang, Zhou Jian. Implementation of Dynamic VPN Based on Openswan[J]. JHUT,2006

[12] 董晓虎，徐明伟，徐恪. 密钥交换协议 IKE 实现的可扩展设计[J]. 小型微型计算机系统，2004,25(6):1000-1004