GOSSIP协议

一. 总述

gossip过程是由种子节点发起，当一个种子节点有状态需要更新到网络中的其他节点时，它会随机的选择周围几个节点散播消息，收到消息的节点也会重复该过程，直至最终网络中所有的节点都收到了消息。这个过程可能需要一定的时间，由于不能保证某个时刻所有节点都收到消息，但是理论上最终所有节点都会收到消息，因此它是一个最终一致性。

> 1. Gossip 是周期性的传播消息，我们可以对传播周期进行设定
> 2. 被传染的节点要随机选择`k`个临近点进行传播，当然这个`k`值也是需要我们自己来进行设定的
> 3. 在传播过程中，一定要注意是往**未发送过的节点**传播的
> 4. 收到消息的节点将不会再继续往发送节点传播，也就是说消息传播都是单向的

二. 优点

1. 扩展性

在网络中可以很容易的进行扩展，节点可以任意的增加和减少

2. 容错

任何一个节点宕机都不会影响到`gossip`的传播，这对分布式系统来说是一个很好的特性

3. 去中心化

`gossip`协议不要求任何中心节点，所有节点都可以是对等的，任何一个节点无需知道整个网络状况，只要网络是连通的，任意一个节点就可以把消息散播到全网。

4. 一致性收敛

传播的方式是一传十，十传百的，速度会越来越快，可以保证在一定时间内实现所有节点的消息都收敛到一致，传播速度也达到了$logN$

三. Gossip模型

反熵模式

`Anti-Entropy`是 **SI model**，节点只有两种状态，`Suspective`和`Infective`，叫做`simple epidemics`。

反熵的含义就是要降低节点之间的混乱程度，而实现节点间的高度相似。

在 **SI model**下，一个节点会把所有的数据都跟其他节点共享，以便消除节点之间数据的任何不一致，它可以保证最终、完全的一致。

由于在 **SI model**下消息会不断反复的交换，因此消息数量是非常庞大的、无限制的，这对一个系统来说是一个巨大的开销。

Rumor-Mongering

这一种称为谣言传播，它仅传播新到达的数据

`Rumor-Mongering`是 **SIR model**，节点有三种状态，`Suspective`，`Infective`和`Removed`，叫做`complex epidemics`。

但是在 **Rumor Mongering (SIR Model)**模型下，消息可以发送得更频繁，因为消息只包含最新 `update`，体积更小。而且，一个 `Rumor`消息在某个时间点之后会被标记为 `removed`，并且不再被传播，因此，**SIR model**下，系统有一定的概率会不一致。

而由于，**SIR Model**下某个时间点之后消息不再传播，因此消息是有限的，系统开销小。

因此，对于每种方法该如何进行选择是需要经过长时间考虑的。

四. 通信模式

在 Gossip 协议下，网络中两个节点之间有三种通信方式:

1. Push: 节点 A 将数据 (`key`,`value`,`version`) 及对应的版本号推送给 B 节点，B 节点更新 A 中比自己新的数据，在`Fabric`的论文中对于这个一点也有较为详细的论述
2. Pull: A 仅将数据 `key`,`version`推送给 B，B 将本地比 A 新的数据 `（Key, value, version）`推送给 A，A 更新本地
3. Push/Pull: 与 `Pull`类似，只是多了一步，A 再将本地比 B 新的数据推送给 B，B 则更新本地

五. 伪算法

```
do forever
  wait(T time units)
  p ← selectPeer()
  if push then
    // 0 is the initial hop count
    myDescriptor ← (myAddress, 0)
    buffer ← merge(view,{myDescriptor})
    send buffer to p
  else
    // empty view to trigger response
    send {} to p
  if pull then
    receive view_p from p
    view_p ← increaseHopCount(view_p)
    buffer ← merge(view_p,view)
    view ← selectView(buffer)
```

(a) active thread

```
do forever
  (p, view_p) ← waitMessage()
  view_p ← increaseHopCount(view_p)
  if pull then
    // 0 is the initial hop count
    myDescriptor ← (myAddress, 0)
    buffer ← merge(view,{myDescriptor})
    send buffer to p
  buffer ← merge(view_p,view)
  view ← selectView(buffer)
```

(b) passive thread

六. 可能面临的攻击模型

1.女巫攻击（Sybil Attack）

> 很多攻击者的同伙加入网络，攻击者一个人支配了全网。

2.日蚀攻击（Eclipse Attack）

> 攻击者串通了正常节点(V)的邻居，使V被孤立，且攻击者能得知V所有的内容。

3.扰动攻击（Churn Attack）

> 由于加入/退出P2P网络，几乎不需要耗费任何资源，攻击者可以频繁地进出网路，使邻近节点疲于更新路由表，造成网路阻塞及离线资源无法取得。

4.敌对路由（Adversarial Routing）

> 在P2P开放的环境下，参与者的行为是不可预期的，攻击者可以无视节点列表而将讯息接力给攻击者的隔离网路或直接拒绝处理。

5.污染攻击（Pollution Attack）

> 污染攻击与敌对路由手法同样地简单而暴力：攻击者直接篡改传播信息，导致全网收到错误数据；

6.服务阻断（Denial of Service）

> 服务阻断就是攻击者透过各种手段耗竭服务提供者有限的资源导致其无法再提供任何服务，以达成瘫痪节点服务的目的。也称为DDoS攻击.

七 参考实现

> ethereum/p2p;
>
> ethereum/whisper;
>
> libp2p/go-pubsub;