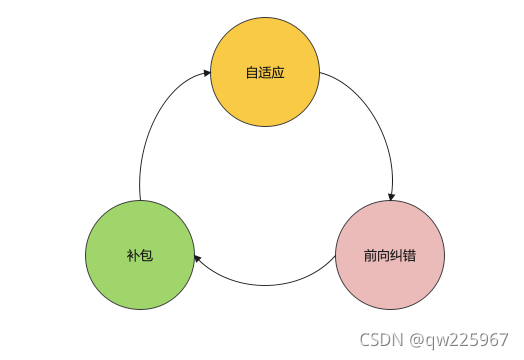
# **一、弱网优化简介**

  目前用户的网络环境相比过去十年已经有了较大的改善。但是针对移动网络以及其他弱网环境，我们仍需想办法让我们的服务做到处变不惊。也就是在任何环境下，都能提供流畅的画面播放（针对音视频传输方面）。  
  目前常使用的是RTP协议来进行音视频的传输，而这个协议是基于UDP的，因此引入了很多保证网络传输质量的模块。针对音视频传输常用的拥塞控制算法有：webrtc中的transport-cc、quic中的bbr等，这两种算法都是基于延迟进行带宽估计的。而媒体数据传输对实时性有极高的要求，同时需要保证数据的连续性。因此不仅要关注传输过程中丢包的情况，还需要应对延迟增大的问题。

  弱网优化这个命题非常大，因为应对弱网的方法是多样的。例如：补包策略（ACK、NACK等）、前向纠错（FEC）、码率自适应等。补包与前向纠错都有各自的优缺点，补包策略是最简单有效的抗弱网方式。  


## **1.1 补包**

  简单来说就是当网络传输时发送丢包，我们立即通知对方进行补发的一种机制。这种方式将会消耗较多的带宽资源。

## **1.2 前向纠错**

  前向纠错的方式则是通过提前封装冗余包来抵抗网络的丢包问题，当对端接收时发生丢包，那么可以根据冗余包的数据进行恢复，实现抗弱网。

## **1.3 自适应**

  自适应方法的关键就是对带宽的预估以及对输出码率的调整，除了要对网络状况有一个准确的估计之外，还需要找到一个合适的码率输出值。我们需要进行大量的实验来保证用户体验，需要考虑是牺牲帧率还是牺牲分辨率来降低码率。

  事实上，所有抗弱网实践中，我们都不可能依靠某一个方法达到最佳效果，因此大部分厂商在实现时所有场景都会用到。下面，我们聚焦在带宽估计的部分，来分析一下quic中的BBR算法在RTC中的应用。

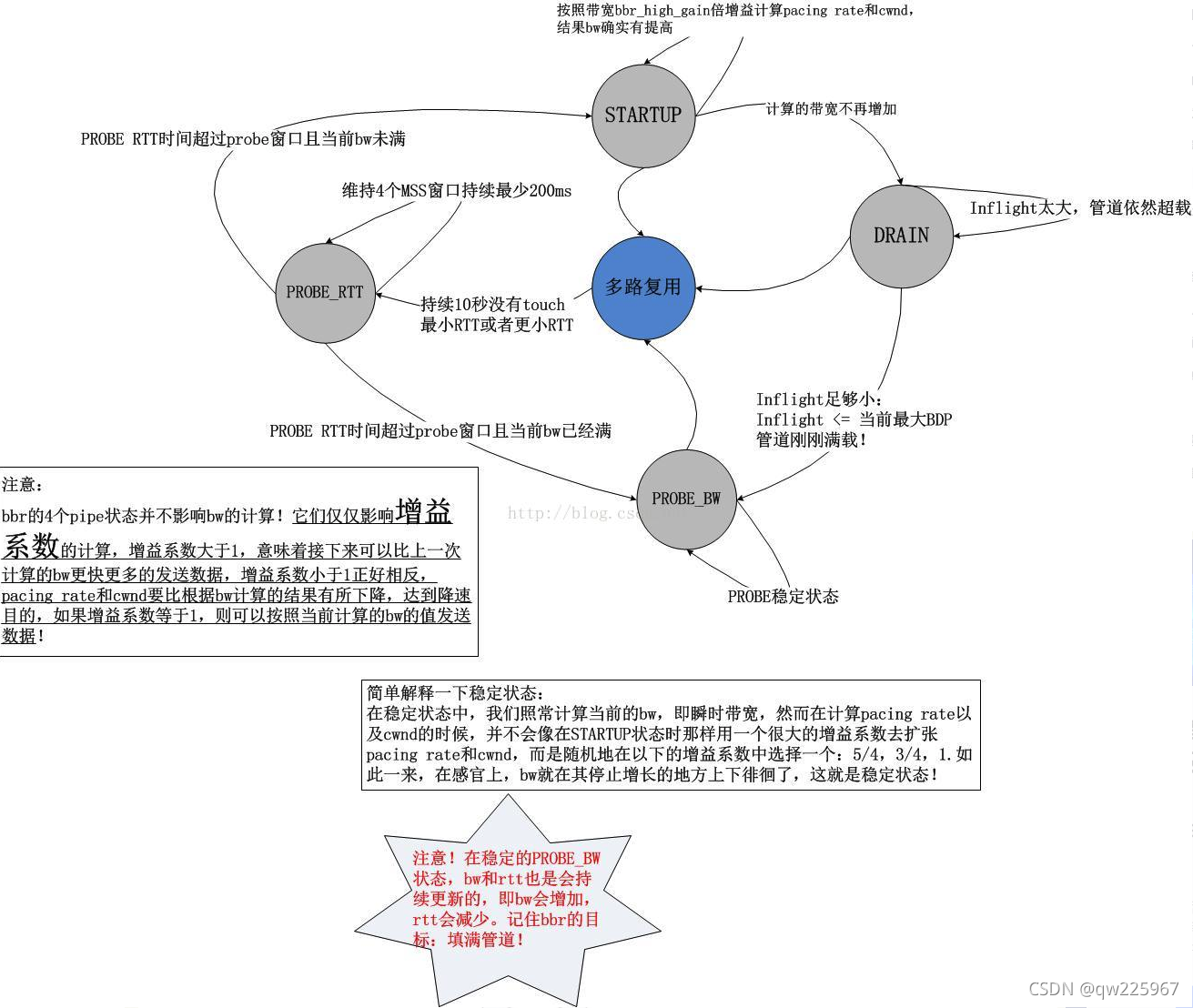
# **二、BBR算法**

  自1980年代，TCP的拥塞控制算法现世到如今，网络环境发生了巨大的变化。原先基于数据损耗的拥塞算法已经无法满足当前的传输控制需求。因此谷歌提出了一种基于带宽瓶颈计算数据往返时间来实现的拥塞控制算法（BBR——Bottleneck Bandwidth and RTT）。该算法通过安全而又直接的带宽检测迅速获得当前链路的带宽值、rtt。同时，一改旧拥塞算法的损耗参考思路，在拥塞发生时BBR算法依然可以插手cwnd的调整，有效解决了硬着陆的问题。

## **2.1 算法组成**

  BBR算法的组成其实非常简单，主要由以下五个部分组成：  
  1.即时速率的计算  
  带宽计算是整个BBR算法的基准，其中即时带宽统计的方法是具有里程碑意义的。它突破旧算法的信令参考，不再关心数据内涵。化繁为简转而关注数据量本身，无论是SACK、RACK、RTO都不再是其关注的重点，使得整个检测流程变得更直接更高效。  
  2.RTT的跟踪  
  RTT与上述的带宽可以有效的描述当前链路的最大容量。BBR算法探测的是RTT的最小值，它无时无刻都在尝试达到这个链路的最大容量，因此展现出了极高的带宽利用率。  
  3.BBR pipe状态机的维持  
  BBR算法根据互联网的拥塞行为有针对性地定义了4中状态，即STARTUP，DRAIN，PROBE\_BW，PROBE\_RTT。BBR通过对上述计算的即时带宽bw以及rtt的持续观察，在这4个状态之间自由切换。  
  4.结果输出-pacing rate和cwnd  
  pacing rate可以控制数据包的发送间隔，在每一时刻都会有一个与之匹配的pacing rate来控制发送，大大提高了整个容量的利用。  
与传统意义的拥塞算法相比，cwnd不再是BBR算法的唯一输出。事实上，更重要的是BBR定义了pacing rate这个参考值。cwnd只能控制发出去多少数据，但无法控制怎么去发送这些数据。因此整个容量存在许多不充分利用的情况甚至会造成路由器排队形成深队列。  
  5.其它外部机制的利用 fq，rack等  
  BBR算法对外部机制进行了高效的利用。例如：使用了FQ中的平缓发送来实现pacing rate等。

## **2.2 状态机**

  以上的五个内容通过BBR内部的四个状态切换来实现动态控制。下图引用网络中对bbr状态机常用的描述图：  


  阶段一（STARTUP），使用最高的增益值来进行发送增长，当计算到带宽不在增加时则进入下一阶段；  
  阶段二（DRAIN），前面的发送速度增长非常快，因此很容易在带宽最高的那一段时间内造成管道超载，因此需要进行一部分的排空操作；  
  阶段三（PROBE\_BW），该状态为稳定状态。当管道排空完毕，带宽正好满足数据传输需求，因此进入了一个传输大小稳定的一个时期；  
  阶段四（PROBE\_RTT），当网络出现拥塞时将会进入该状态，判定条件为：RTT时间超过probe窗口而且当前bw已满或者在rtt更新周期内没有发现更小的rtt。这个状态将会减少发送数量，达到排空的目的。

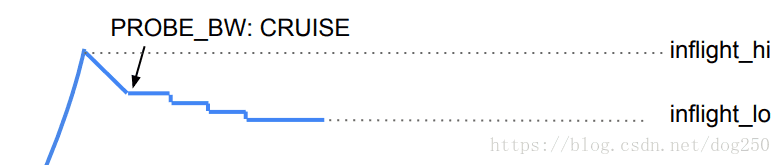
## **2.3 BBR优劣分析**

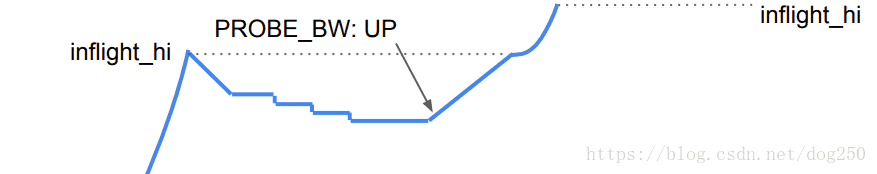
  我们刚刚讨论的都是bbr的第一个版本，近期bbr已经更新了它的v2版本，但是这里还是先介绍v1版本中的缺陷。

  在BBRv1版本的实践中发现，该算法存在以下几个缺陷（TCP应用相关）：  
  1.过于激进的启动操作：  
由于启动阶段探测的容积较大，导致发送速率以及数量偏高。这会影响正在传输的TCP流，形成深队列，造成链路中传输拥塞。[1]  
  2.过于激进的带宽探测：  
当整个链路存在多个BBRv1算法的TCP流时，BBR会高估整个链路的传输容量，导致每个TCP流形成竞争造成大量丢包。[2]  
  3.不公平的RTT探测：  
当一个链路中存在两个BBRv1的流，其中一个流的RTT大于另一个流的RTT时。计算出来的BDP相差较大，RTT较大的流会占用更多的带宽，形成整个传输的不公平性。[2]  
  4.损耗型算法的不公平竞争：  
此处举例一个比较好理解的例子：例如当传统基于损耗的拥塞算法与BBRv1同时共享一个窄带通道。当发生拥塞时，虽然BBRv1检测到了带宽下降但是无法检测到拥塞产生。相反传统的拥塞算法会检测丢包并强行降低发包，这导致了BBRv1算法的非法竞争。

  针对上面的缺陷，BBRv2算法于 IETF-102 和 IETF-104进行了改进：  
  1.缓解了Startup阶段以及激进发送带来的丢包和时延；  
  2.改善了与传统CC算法并存时的公平性问题；  
  3.重构了PROBE\_RTT这个BBR状态的实现。

### **2.3.1 PROBE\_BW重构**

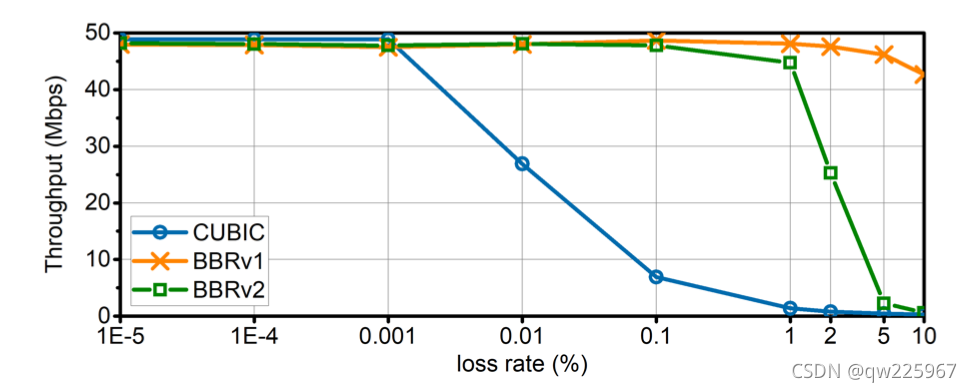
  PROBE\_BW变化体现在三方面（引用dog250大佬的解释：https://blog.csdn.net/dog250/article/details/81141638）：  
  Cruise：不再像BBR v1.0那般保持estimateBDP\*1gain这样，而是确保inflight在inflight\_hi之下保留一定的空间。  


  ProbeMore：采用了类似SlowStart或者Startup阶段的指数级增加探测包的策略，同时在丢包时设置inflight\_hi…  


  ProbeDrain：由于BBR v1.0收敛实在太慢，BBR v2.0采用了一次性收敛的策略。

### **2.3.2 优势**

#### **抗丢包性能**

  其实稍微了解bbr的朋友可能都知道，bbr算法只是计算即时带宽并不会对丢包进行参考，那么何来的抗丢包性能呢？这里的抗丢包性能可以这么理解。例如：当网络中存在5%的丢包时，大部分的数据往返时间都是确定的，那么总能计算出一个复合算法要求的最小rtt，那么整体的rtt不会发送太大的变化，因此估计出来的带宽不会有太大的影响。当网络中存在较大的丢包（>5%）时，那么我们计算的rtt将会由于中间丢包造成影响，因此带宽会出现所谓的断崖式下跌的情况：  


#### **低延迟与高抢占**

  我们可知在所有的TCP拥塞算法中，BBR都具备极高的抢占能力，这是因为其启动阶段的超高增益导致的。BBR算法的思想是以最短的时间去探测到带宽的最大值，以便时刻维持在PROBE\_BW这样的状态下。而且，即使在稳定阶段，BBR仍然会根据周期变化来上调我们的发送增益，不断试探是否存在更多的带宽。这种方式带来的优势是：数据包可以尽可能的发送到网络中，保证了低延迟以及高抢占。

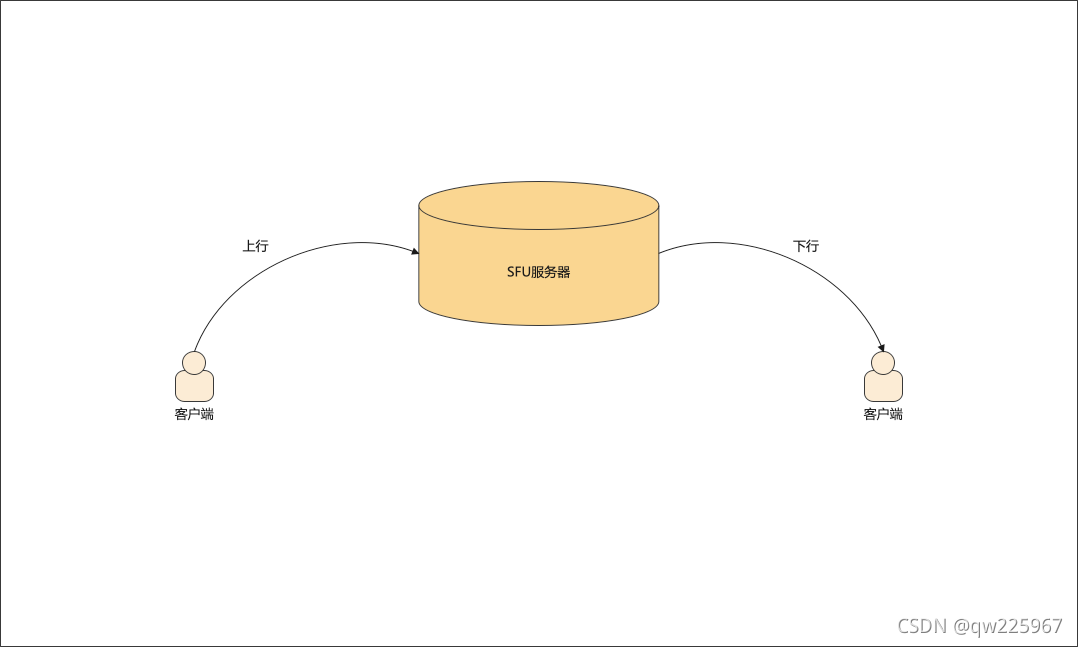
#### **准确的带宽反馈**

  BBR算法使用的是及时带宽，也就是说BBR永远不去做估计，只是反应当前的最大带宽情况。这样的方式比普通的带宽估计更准确，更可靠。

# **三、应用**

  经过上述的介绍，我们基本了解了BBR的算法基础以及优劣，下面我介绍一些对其在RTC中应用的尝试。

## **3.1 RTC场景**

  目前，大部分的RTC场景都不是WebRTC那样的P2P模式，而是存在转发服务器的SFU场景。这类场景无法像P2P场景那样直接在双方的交互上做工作，而是要考虑上行场景以及下行场景的不同。  


（客户端 ——> 服务器），这条上行链路上，我们客户端可以做的事情比较多。  
  上行网络异常，客户端可以进行码率自适应、FEC、NACK等方式去适配我们的上行弱网情况。  
（服务器 ——> 客户端）在下行链路时。  
  服务器为了保证实时性，不会对数据包进行编解码操作，因此是无法实现无损降码率的。这两种场景下，我们在带宽算法的选择上就出现了差异。

### **音视频混合场景**

  撇开上行网络不谈，我们强调解决下行网络的弱网情况，那么可以明确的是，下行网络中必定会使用到丢包补偿策略。因为丢包补偿策略不需要调整码率，只是简单的向服务器要求重传即可。但是这样单一的方式则会造成以下缺陷：带宽受限时传输直接崩溃。因为在带宽受限时下行的客户端并不知道网络传输受限，因此不断要求重传导致了进一步加重网络拥塞，直到完全崩溃。因此在下行服务器部分引入一个带宽估计算法是必不可少的。  
  当前场景我们定位为音视频混合的场景，可知音频数据占用带宽与视频数据占用带宽相比小很多，同时音频数据粘性比视频要低（这里理解为视频关键帧数据需要多个数据包组成一个帧，任何一个包丢失都会造成数据失效，因此粘性高）。在此我们首先要保证在整个传输链路中占比最大的、粘性最高的数据要在我们尽量可控范围内进行传输。因此我们针对视频数据进行了带宽估计以及发送控制。

  BBR带宽估计算法在下行网络中的可行性：  
  1.即时下行带宽测算：  
  我们下行网络能做的，就是对下行网络进行准确的带宽统计并反馈给上行去进行码率调整，实现下行抗弱网，BBR的带宽统计比其他的算法更符合实际带宽情况；  
  2.控制发送速率：  
  在反馈给上行的同时，还需要对下行发送数据的控制，把我们发送到网络的数据实现可控。这里不但要把正常的传输数据包进行限制，连带重传的数据包也需要进行进一步的控制；  
  3.激进的带宽算法：  
  下行网络与上行不同，目前家用下行网络普遍带宽高于上行，因此我们需要在拥塞解除后立刻进行带宽抢占，而BBRV1的激进做法非常适合。  
  4.音频抢占问题：  
  BBR算法提供了比较强力的带宽抢占能力，因此在与裸流的竞争中不至于过于被动，而我的音频流为无控制的数据流，因此BBR相比来说优势更大。

## **3.2 实现思考**

  在RTC实现中，我们优先考虑到数据转发的性能影响，因此大部分的SFU服务器都是多进程——单线程模式，因此在BBR的计算中不能引入过多的时间消耗，从而影响整体的传输效率。  
  在使用RTP协议进行传输时，我们为了保证实时性时长使用的NACK这样的补包策略——也就是在出现网络丢包时再通知对方补发而不是像TCP协议中持续的ACK来保证数据的可靠性。而这里引入的quic协议中的BBR则依赖于持续的ACK来实现的，因此在实现中做了大量的适配。  
  1.ack策略改为100ms内统一回复这段时间内接到的包以及数据停留时长。  
  这样的方式不会引入过于频繁的确认操作，但是会引入一些确认问题：当ack信令丢失时会出现大量的未确认情况、同时确认滞后。而且RTT的计算需要在服务端进行，RTT = 当前的确认时间 - 停留时间 - 发送时间；  
  2.反馈策略为200ms对上行进行一次带宽反馈。  
  BBR的带宽估计下降非常迅速，因此我们不但在反馈上行时故意调低了反馈带宽（真实带宽 \* 0.9）来应对波动，同时还加入了物理丢包补偿，实现在400ms内数据迅速准确下降；  
  3.物理丢包补偿与补包后丢包参考  
  在实现中，我们加入了两个丢包参考量。我们都知道BBR在抗丢包上几乎是毫无措施，因此我们需要在外层做丢包策略。首先是补包后丢包：意思是经过重传后的丢包率，实际反映了下行客户端有效数据接收量。这里我们引入了google的丢包带宽估计算法，应对带宽稳定时额外的丢包率问题；其次是物理丢包率，我们同时还统计了下行接收的情况以及上行发出的情况，来计算准确的物理丢包率，这样可以准确的添加到BBR的增益中，同时反馈上行时故意降低发送带宽，为我们下行重传腾出足够的带宽空间。  
  4.发送数据策略  
  首先发送数据是一个比较头疼的问题，因为BBR输出的Pacing\_Rate是一个带宽值，也就意味是两个变量组成：数据量和时间。在发送时我们要怎么去界定是控制发送间隔还是控制发送数据量的多少呢？我们先解释两种做法的不同。首先，控制发送间隔调整数据量，这样的做法很有可能会导致我们发包的数据不够均匀，同时当网络极差的时候可能会有多数间隔会无法发送数据。其次，控制发包数据调整间隔，这样的做法则是在发送时无法可靠控制发送间隔，导致下行接包忽快忽慢，最致命的是在网络剧烈抖动时间隔相差可能很大，不利于下行处理。因此我们此处把发送间隔根据RTT的状态进行切换，同时在切换的过程中根据对应的间隔去调整发包量，这样在稳定的间隔内控制发包数据则会更加平滑。不仅如此，我们还会让每一个间隔内必须要发送一个数据包，即使现在的网络已经较差，这样的做法是为了保证每一个间隔都有数据流出，不至于激增我们的延迟。  
  5.丢帧策略  
  在应对1v1场景时，我们可以肆无忌惮的反馈给上行去调整码率，但是在多人的下行网络中，我们是不可能因为其中一人的问题而牺牲所有人的体验，因此，多人部分在服务端降码率的方式只能是把不重要的帧丢掉而实现降码率。这样的方式是要结合自身的业务场景，尽量保证I帧的发送，把不必要的P/B帧丢掉来降低码率。

# **四、总结**

  上面对常用的抗弱网方式进行了简介，同时着重介绍了BBR拥塞控制算法，最后在简单总结了近期使用BBR算法来实现下行弱网优化的关键点，以后有时间将会继续拓展这部分的内容。