# Modelling the Congestion Control Problem

We treat congestion control as a problem of distributed decision making under uncertainty. Each endpoint that has pending data must decide for itself at every instant: send a packet, or don’t send a packet.

我们将拥塞控制视为不确定性下的分布式决策问题。每个具有待处理数据的节点都必须在每个瞬间自行决定：发送数据包，还是不发送数据包。

If all nodes knew in advance the network topology and capacity, and the schedule of each node’s present and future offered load, such decisions could in principle be made perfectly, to achieve a desired allocation of throughput on shared links.

如果所有节点都事先了解网络拓扑结构和容量以及每个节点当前和将来提供的负载的时间表，则原则上可以完美地做出这样的决定，以在共享链路上实现所需的吞吐量分配。

In practice, however, endpoints receive observations that only hint at this information. These include feedback from receivers concerning the timing of packets that arrived and detection of packets that didn’t, and sometimes signals, such as ECN marks, from within the network itself. Nodes then make sending decisions based on this partial information about the network.

但是，实际上，节点会收到仅暗示此信息的观察结果。其中包括来自接收者的反馈，涉及到达的数据包的时间安排和检测未到达的数据包的反馈，有时还包括来自网络本身内部的信号，例如ECN标记。然后，节点根据有关网络的部分信息做出发送决策。

Our approach hinges on being able to evaluate quantitatively the merit of any particular congestion control algorithm, and search for the best algorithm for a given network model and objective function. We discuss here our models of the network and cross traffic, and how we ultimately calculate a figure of merit for an arbitrary congestion control algorithm.

我们的方法取决于能否定量评估任何特定拥塞控制算法的优点，并为给定的网络模型和目标函数寻找最佳算法。我们在这里讨论我们的网络和交叉流量模型，以及我们最终如何为任意拥塞控制算法计算品质因数。

3.1 Expressing prior assumptions about the network

From a node’s perspective, we treat the network as having been drawn from a stochastic generative process. We assume the network is Markovian, meaning that it is described by some state (e.g. the packets in each queue) and its future evolution will depend only on the current state.

从节点的角度来看，我们将网络视为是从随机生成过程中提取的。我们假设网络是马尔可夫网络，这意味着它由某种状态（例如每个队列中的数据包）描述，并且其未来的发展将仅取决于当前状态。

Currently, we typically parameterise networks on three axes: the speed of bottleneck links, the propagation delay of the network paths, and the degree of multiplexing, i.e., the number of senders contending for each bottleneck link. We assume that senders have no control over the paths taken by their packets to the receiver.

当前，我们通常在三个轴上对网络进行参数设置：1) 瓶颈链路的速度，2) 网络路径的传播延迟以及3) 多路复用的程度，即每个瓶颈链路的发送者数量。我们假设发送方无法控制其数据包到达接收方的路径。

Depending on the range of networks over which the protocol is intended to be used, a node may have more or less uncertainty about the network’s key parameters. For example, in a data center, the topology, link speeds, and minimum roundtrip times may be known in advance, but the degree of multiplexing could vary over a large range. A virtual private network between “clouds” may have more uncertainty about the link speed. A wireless network path may experience less multiplexing, but a large range of transmission rates and roundtrip times.

根据打算使用该协议的网络范围，一个节点可能对网络的关键参数具有或多或少的不确定性。例如，在数据中心中，拓扑，链接速度和最小往返时间可能是预先已知的，但是多路复用的程度可能会在较大范围内变化。“云”之间的虚拟专用网络可能对链接速度有更多的不确定性。无线网络路径可能经历较少的复用，但是传输速率和往返时间的范围很大。

As one might expect, we have observed a trade-off between generality and performance; a protocol designed for a broad range of networks may be beaten by a protocol that has been supplied with more specific and accurate prior knowledge. Our approach allows protocol designers to measure this trade-off and choose an appropriate design range for their applications.

正如人们可能期望的那样，**我们已经观察到通用性和性能之间的权衡**； 为多种网络设计的协议可能会被提供了更具体，准确的先验知识的协议所击败。我们的方法允许协议设计者衡量这一折衷，并为其应用选择合适的设计范围。

3.2 Traffic model

Remy models the offered load as a stochastic process that switches unicast flows between sender-receiver pairs on or off. In a simple model, each endpoint has traffic independent of the other endpoints. The sender is “off” for some number of seconds, drawn from an exponential distribution. Then it switches on for some number of bytes to be transmitted, drawn from an empirical distribution of flow sizes or a closed-form distribution (e.g. heavy tailed Pareto). While “on,” we assume that the sender will not stall until it completes its transfer.

Remy将提供的负载建模为一个随机过程，该过程可以打开或关闭发送方-接收方对之间的单播流。在简单模型中，每个节点的流量都独立于其他节点。根据指数分布，发送方处于“关闭”状态数秒钟。然后，它根据流大小的经验分布或封闭形式的分布（例如，重尾的帕累托）打开一些要传输的字节。当“打开”时，**我们假设发送者直到完成传输才停止**。

In traffic models characteristic of data center usage, the offtoon switches of contending flows may cluster near one another in time, leading to in-cast. We also model the case where senders are “on” for some amount of time (as opposed to bytes) and seek maximum throughput, as in the case of videoconferences or similar real-time traffic.

在以数据中心使用为特征的流量模型中，竞争流的尾部开关可能会在时间上彼此靠近，从而导致播出。我们还对以下情况进行了建模：发送者“打开”了一段时间（而不是字节），并寻求最大吞吐量，例如在视频会议或类似的实时流量中。

3.3 Objective function

Resource allocation theories of congestion control have traditionally employed the alpha-fairness metric to evaluate allocations of throughput on shared links [37]. A flow that receives steady-state throughput of x is assigned a score of Ua (x) = 1x1 aa . As a ! 1, in the limit U1(x) becomes log x.

拥塞控制的资源分配理论传统上采用alpha-公平性度量来评估共享链路上的吞吐量分配[37]。为接收x的稳态吞吐量的流分配的分数。当，极限是。

Because Ua (x) is concave for a > 0 and monotonically increasing, an allocation that maximizes the total score will prefer to divide the throughput of a bottleneck link equally between flows. When this is impossible, the parameter a sets the trade-off between fairness and efficiency. For example, a = 0 assigns no value to fairness and simply measures total throughput. a = 1 is known as proportional fairness, because it will cut one user’s allocation in half as long as another user’s can be more than doubled. a = 2 corresponds to minimum potential delay fairness, where the score goes as the negative inverse of throughput; this metric seeks to minimize the total time of fixed length file transfers. As , maximizing the total Ua (x) achieves max-min fairness, where all that matters is the minimum resource allocations in bottom-up order [37].

因为对于呈凹形并且单调增加，所以使总得分最大化的分配将更偏向在流之间平均分配瓶颈链路的带宽。如果这不可实现，则参数*α*设置公平性和效率之间的权衡。例如，并不能调整流的公平，而只是测量总吞吐量。为比例公平，因为只要另一个用户的分配可以增加一倍以上，它将把一个用户的分配减少一半。a = 2对应于最小潜在延迟公平性，其中分数与吞吐量的负数成正比； 此度量旨在最大程度地减少固定长度文件传输的总时间。令，最大化就可以实现最大-最小公平性，其核心是自下而上的顺序中的最小资源分配[37]。

Because the overall score is simply a sum of monotonically increasing functions of throughput, an algorithm that maximizes this total is Pareto-efficient for any value of *α*; i.e., the metric will always prefer an allocation that helps one user and leaves all other users the same or better. Tan et al. [28] proved that, subject to the requirement of Pareto-efficiency, alpha-fairness is the metric that places the greatest emphasis on fairness for a particular *α*.

因为总得分只是吞吐量单调增加函数的总和，所以对于任何α值，使总和最大化的算法是帕累托有效的； 即，该指标将始终偏向于一种分配，该分配可以帮助一个用户，并使所有其他用户保持相同或更好。Tan等。[28] 证明，在帕累托效率的要求下，α公平是最强调特定α公平的度量。

Kelly et al. [25] and further analyses showed that TCP approximately maximizes minimum potential delay fairness asymptotically in steady state, if all losses are congestive and link speeds are fixed.

Kelly等。[25] 与进一步的分析表明，如果所有损耗都是拥塞且链路速度是固定的，则TCP会在渐近状态下渐近渐近最大化最小潜在延迟公平性。

We extend this model to cover dynamic traffic and network conditions. Given a network trace, we calculate the average throughput x of each flow, defined as the total number of bytes received divided by the time that the sender was “on.” We calculate the average roundtrip delay y of the connection.

我们将此模型扩展到涵盖动态流量和网络条件。给定网络痕迹，我们计算每个流的平均吞吐量*x*，*x*定义为接收到的字节总数除以发送者“打开”时间。我们计算连接的平均往返延迟*y*。

The flow’s score is then 

where *α* and *β* express the fairness efficiency trade-off in throughput and delay, respectively, and *δ* expresses the relative importance.

其中α和β分别表示吞吐量和延迟之间的公平效率权衡，而δ表示相对重要性。

We emphasize that the purpose of the objective function is to supply a quantitative goal from a protocol-design perspective. It need not (indeed, does not) precisely represent users’ “true” preferences or utilities. In real usage, different users may have different objectives; a videoconference may not benefit from more throughput, or some packets may be more important than others. We have not yet addressed the problem of how to accommodate diverse objectives or how endpoints might learn about the differing preferences of other endpoints.

我们强调目标功能的目的是从协议设计的角度提供量化目标。它不一定（实际上也不必）准确地表示用户的“真实”偏好或实用程序。在实际使用中，不同的用户可能有不同的目标。视频会议可能无法从更多的吞吐量中受益，或者某些数据包可能比其他数据包更重要。我们尚未解决如何适应各种目标或节点如何了解其他节点的不同偏好的问题。

# How Remy Produces a Congestion Control Algorithm

The above model may be viewed as a cooperative game that endpoints play. Given packets to transmit (offered load) at an endpoint, the endpoint must decide when to send packets in order to maximize its own objective function. With a particular congestion control algorithm running on each endpoint, we can calculate each endpoint’s expected score.

可以将以上模型视为节点玩的合作游戏。给定要在节点传输的数据包（提供的负载），节点必须决定**何时**发送数据包，以最大化其自身的目标效用。通过在每个节点上运行特定的拥塞控制算法，我们可以计算每个节点的预期分数。

In the traditional game-theoretic framework, an endpoint’s decision to send or abstain can be evaluated after fixing the behaviour of all other endpoints. An endpoint makes a “rational” decision to send if doing so would improve its expected score, compared with abstaining.

在传统的博弈论框架中，可以在假定所有其他节点的行为后，评估节点的发送或弃权决定。与弃权相比，节点做出“理性”的决定，如“发送”是否可以提高其预期分数。

Unfortunately, when greater individual throughput is the desired objective, on a best-effort packet switched network like the Internet, it is always advantageous to send a packet. In this setting, if every endpoint acted rationally in its own self-interest, the resulting Nash equilibrium would be congestion collapse! [[1]](#footnote-1)This answer is unsatisfactory from a protocol design perspective, when endpoints have the freedom to send packets when they choose, but the designer wishes to achieve an efficient and equitable allocation of network capacity.

不幸的是，当更大的吞吐量是理想的目标时，在尽力而为的分组交换网络上，例如Internet，发送分组总是有利的。在这种情况下，如果每个节点都出于自己的利益采取合理的行动，则纳什均衡将导致拥塞崩溃！从协议设计的角度看，当节点可以自由选择发送数据包时，这个答案是不令人满意的。设计者希望实现网络容量的有效和公平分配。

Instead, we believe the appropriate framework is that of super-rationality [20]. Instead of fixing the other endpoints’ actions before deciding how to maximize one endpoint’s expected score, what is fixed is the common (but as yet unknown) algorithm run by all endpoints. As in traditional game theory, the endpoint’s goal remains maximizing its own self-interest, but with the knowledge that other endpoints are reasoning the same way and will therefore arrive at the same algorithm.

相反，我们认为适当的框架是超理性的框架[20]。即在最大化一个节点的预期分数时，不假定其他节点的行为，而是假定的是所有节点运行的通用算法（但尚不清楚）。与传统博弈论一样，节点的目标仍然是最大化自身利益，同时假定其他节点会以相同的方式推理，因此将得出相同的算法。

Remy’s job is to find what that algorithm should be. We refer to a particular Remy designed congestion control algorithm as a “RemyCC,” which we then implant into an existing sender as part of TCP, DCCP [26], congestion manager [5], or another module running congestion control. The receiver is unchanged (as of now; this may change in the future), but is expected to send periodic ACK feedback.

Remy的工作是找到那种算法。我们将特定的Remy设计的拥塞控制算法称为“ RemyCC”，然后将其植入到现有的发送方中，作为TCP，DCCP [26]，拥塞管理器[5]或其他运行拥塞控制的模块的一部分。接收器保持不变（到目前为止；这可能会在将来改变），但是有望发送周期性的ACK反馈。

Formally, we treat the problem of finding the best RemyCC under uncertain network conditions as a search for the best policy for a decentralized partially observable Markov decision process, or DecPOMDP [34]. This model originated from operations research and artificial intelligence, in settings where independent agents work cooperatively to achieve some goal. In the case of end-to-end congestion control, endpoints are connected to a shared network that evolves in Markovian fashion. At every time step, the agents must choose between the actions of “sending” or “abstaining,” using observables from their receiver or from network infrastructure.

形式上，我们将在不确定的网络条件下寻找最佳RemyCC的问题视为“对分散的，部分可观察的马尔可夫决策过程或**分布式部分可见马尔科夫决策过程**进行决策的最佳策略[34] ”。该模型起源于运筹学和人工智能，在这种情况下，独立代理协同工作以实现某些目标。在端到端拥塞控制的情况下，端点连接到以马尔可夫方式发展的共享网络。在每个时间步骤中，代理都必须使用其接收者或网络基础结构中的观察对象在“发送”或“弃权”操作之间进行选择。

4.1 Compactly representing the sender’s state

In principle, for any given network, there is an optimal congestion control scheme that maximizes the expected total of the endpoints’ objective functions. Such an algorithm would relate (1) the entire history of observations seen thus far (e.g. the contents and timing of every ACK) and (2) the entire history of packets already sent, to the best action at any given moment between sending a new packet or abstaining. However, the search for such an algorithm is likely intractable; on a general DecPOMDP it is NEXPcomplete [8].

原则上，对于任何给定的网络，都有一个最佳的拥塞控制方案，可以使端点的目标功能的预期总数最大化。这样的算法将（1）到目前为止所观察到的整个历史（例如每个ACK的内容和定时）和（2）已发送的数据包的整个历史与发送新的两次之间的任何给定时刻的最佳动作相关。包装或弃权。然而，寻找这样的算法可能很棘手。在一般的DPOMDP上，它是NEXPcomplete [8]。

Instead, we approximate the solution by greatly abridging the sender’s state. A RemyCC tracks just three state variables, which it updates each time it receives a new acknowledgment:

相反，我们通过大大简化发送者的状态来近似解决方案。RemyCC仅跟踪三个状态变量，每次收到新的确认时都会对其进行更新：

1. An exponentially-weighted moving average (EWMA) of the inter-arrival time between new acknowledgments received (ack\_ewma).

1.收到的新确认之间的到**达间隔时间**的指数加权移动平均值（EWMA）（ack\_ewma）。

2. An exponentially weighted moving average of the time between TCP sender timestamps reflected in those acknowledgments (send\_ewma). A weight of 1/8 is given to the new sample in both EWMAs.

2.在这些确认（send\_ewma）中反映的TCP发送方时间戳之间的时间的指数加权移动平均值。在两个EWMA中，新样本的权重均为1/8。

3. The ratio between the most recent RTT and the minimum RTT seen during the current connection (rtt\_ratio).

3.在当前连接期间看到的最新RTT与最小RTT之比（rtt\_ratio）。

Together, we call these three variables the RemyCC memory. It is worth reflecting on these variables, which are the “congestion signals” used by any RemyCC. We narrowed the memory to this set after examining and discarding quantities like the most recent RTT sample, the smoothed RTT estimate, and the difference between the long-term EWMA and short-term EWMA of the observed packet rate or RTT. In our experiments, adding extra state variables didn’t improve the performance of the resulting protocol, and each additional dimension slows down the design procedure considerably. But we don’t claim that Remy’s three state variables are the only set that works, or that they are necessarily optimal for all situations a protocol might encounter. We expect that any group of estimates that roughly summarizes the recent history could form the basis of a workable congestion control scheme.

我们一起将这三个变量称为RemyCC内存。可以理解为RemyCC使用的“拥塞信号”。通过检查，我们丢弃了如“最近的RTT样本”，“平滑的RTT估计值”以及“观察到的包速率或RTT的**长期EWMA**和**短期EWMA**之间的差异”。通过我们的实验中，**添加额外的状态变量并不能改善所生成协议的性能**，并且每增加一个维度都会大大降低设计过程的速度。但是，我们并不能肯定Remy的三个状态变量是唯一起作用的变量，或者它们不一定适合协议可能遇到的所有情况。我们希望任何能粗略总结近期历史的估算值都可以构成可行的拥塞控制方案的基础。

We note that a RemyCC’s memory does not include the two factors that traditional TCP congestion control schemes use: packet loss and RTT. This omission is intentional: a RemyCC that functions well will see few congestive losses, because its objective function will discourage building up queues (bloating buffers will decrease a flow’s score). Moreover, avoiding packet loss as a congestion signal allows the protocol to robustly handle stochastic (non-congestive) packet losses without adversely reducing performance. We avoid giving the sender access to the RTT (as opposed to the RTT ratio), because we do not want it to learn different behaviours for different RTTs.

我们注意到，RemyCC的内存不包括传统TCP拥塞控制方案使用的两个因素：丢包和RTT。这种疏忽是有意为之的：运行良好的RemyCC不会造成充血的损失，因为它的目标函数会阻止建立队列（过大的缓冲区会提高惩罚）。此外，避免将分组丢失作为拥塞信号使该协议能够稳固地处理随机（非拥塞）分组丢失，而不会不利地降低性能。我们避免让发送者访问RTT（而不是RTT比率），因为我们不希望它为不同的RTT学习不同的行为。

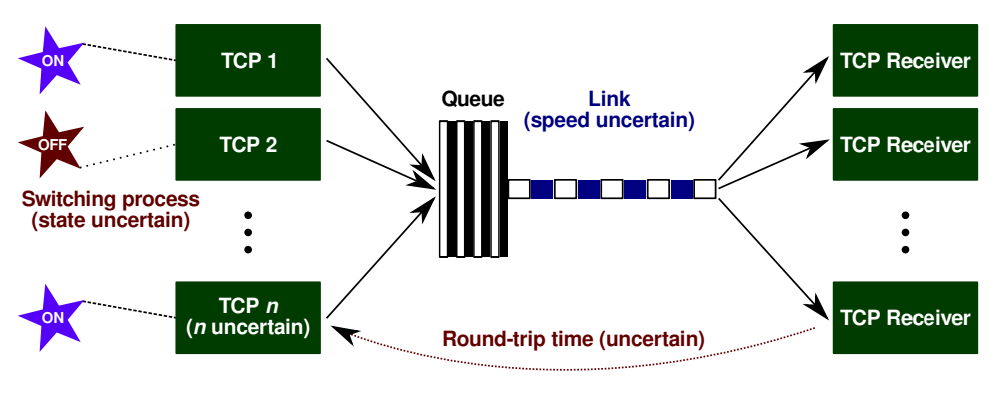


Figure 2: Dumbbell network with uncertainty.

At the start of each flow, before any ACKs have been received, the memory starts in a well-known all zeroes initial state. RemyCCs do not keep state from one “on” period to the next, mimicking TCP’s behaviour in beginning with slow start every time a new connection is established (it is possible that caching congestion state is a good idea on some paths, but we don’t consider this here). Although RemyCCs do not depend on loss as a congestion signal, they do inherit the loss recovery behaviour of whatever TCP sender they are added to.

在每个流的开始，在接收到任何ACK之前，Remy内储以众所周知的全零初始状态启动。RemyCC不会将状态从一个“打开”期间一直保持到下一个，而是模仿每次建立新连接时从缓慢启动开始的TCP行为（在某些路径上缓存拥塞状态可能是个好主意，但是我们不这样做）。尽管RemyCC并不将丢失视为拥塞信号，但它们确实继承了添加到其上的任何TCP发送方的丢失恢复行为。

4.2 RemyCC: Mapping the memory to an action

A RemyCC is defined by how it maps values of the memory to output actions. Operationally, a RemyCC runs as a sequence of lookups triggered by incoming ACKs. (The triggering by ACKs is inspired by TCP’s ACK clocking.) Each time a RemyCC sender receives an ACK, it updates its memory and then looks up the corresponding action. It is Remy’s job to pre-compute this lookup table during the design phase, by finding the mapping that maximizes the expected value of the objective function, with the expectation taken over the network model.

RemyCC通过将内存的值映射到输出动作的方式来定义。在操作上，RemyCC可以理解为由传入ACK触发的一系列查找行为。（ACK的触发灵感来自TCP的ACK时钟。）每次RemyCC发送方收到ACK时，它都会更新其内存，然后查找相应的操作。Remy的工作是在设计阶段通过查找最大化目标函数期望值的映射并在网络模型上获得期望，来预先计算该查找表。

Currently, a Remy action has three components:

1. A multiple m ≥ 0 to the current congestion window (cwnd).

2. An increment b to the congestion window (b could be negative).

3. A lower bound r > 0 milliseconds on the time between successive sends.

当前，Remy动作包含三个部分：

1.当前拥塞窗口（cwnd）的m ≥ 0的倍数。

2.拥塞窗口的增量b（b可以为负）。

3.连续发送之间的时间的下限，r > 0毫秒。

If the number of outstanding packets is greater than cwnd, the sender will transmit segments to close the window, but no faster than one segment every r milliseconds.

如果未完成的数据包数量大于cwnd，则发送方将发送段以关闭窗口，但每*r*毫秒内的传输速度不超过一个段。

A RemyCC is defined by a set of piecewise constant rules, each one mapping a three-dimensional rectangular region of the three-dimensional memory space to a three-dimensional action:

RemyCC由一组分段常量规则定义，每个规则将三维存储空间的三维矩形区域映射到三维动作：



## Remy’s automated design procedure

The design phase of Remy is an optimization procedure to efficiently construct this state-to-action mapping, or rule table. Remy uses simulation of the senders on various sample networks drawn from the network model, with parameters drawn within the ranges of the supplied prior assumptions. These parameters include the link rates, delays, the number of sources, and the on-off distributions of the sources. Offline, Remy evaluates candidate algorithms on millions of randomly generated network configurations. Because of the high speed of current computers and the “embarrassingly parallel” nature of the task, Remy is able to generate congestion control algorithms within a few hours.

Remy的设计阶段是优化过程，**可以有效地构建此状态到动作映射或规则表**。Remy使用从网络模型中提取的各种（由网络上的发送者生成）样本进行仿真，并在提供的先前假设范围内绘制参数。这些参数包括链接速率，延迟，源数以及源的开-关分布。Remy使用**离线方法**评估了数百万个随机生成的网络配置的候选算法。由于当前计算机的高速运行和任务的“令人尴尬的并行”性质，Remy能够在几个小时内生成拥塞控制算法。

A single evaluation step, the innermost loop of Remy’s design process, consists of drawing 16 or more network specimens from the network model, then simulating the RemyCC algorithm at each sender for 100 seconds on each network specimen. At the end of the simulation, the objective function for each sender, given by Equation 1, is totaled to produce an overall figure of merit for the RemyCC. We explore two cases, a = b = 1 and a = 2; d = 0. The first case corresponds to proportional throughput and delay fairness, maximizing

单个评估步骤是Remy设计过程的最深层循环，包括从网络模型中提取16个或更多网络标本，然后在每个发送者上对每个网络标本上的RemyCC算法进行仿真100秒钟。在模拟结束时，由等式1给出的每个发送者的目标函数被总计，以产生RemyCC的总体品质因数。我们探索两种情况，a = b = 1和a = 2； d =0。第一种情况对应于比例吞吐量和延迟公平性，从而使



with δ specifying the importance placed on delay vs. throughput. The second case corresponds to minimizing the potential delay of a fixed length transfer, by maximizing throughput.

δ指定了延迟与吞吐量之间的关系。第二种情况对应于通过最大化吞吐量来最小化固定长度传输的潜在延迟。



Remy initializes a RemyCC with only a single rule. Any values of the three state variables (between 0 and 16,384) are mapped to a default action where m = 1, b = 1, r = 0.01.

Remy仅用一条规则来初始化RemyCC。三个状态变量的任何值（0到16,384之间）都映射到默认操作，其中m = 1，b = 1，r = 0.01。

Each entry in the rule table has an “epoch.” Remy maintains a global epoch number, initialized to 0. Remy’s search for the “best” RemyCC given a network model is a series of greedy steps to build and improve the rule table:

规则表中的每个条目都有一个“周期”。Remy维护一个全局周期序号，并将其初始化为0。给定网络模型后，Remy搜索“最佳” RemyCC是一系列贪婪的步骤，用于构建和改进规则表：

1. Set all rules to the current epoch.

1.将所有规则设置为当前周期。

2. Find the most used rule in this epoch: Simulate the current RemyCC and see which rule in the current epoch receives the most use. If no such rules were used, go to step 4.

2.在此周期找到最常用的规则：模拟当前的RemyCC，查看当前周期中哪个规则使用最多。如果未使用此类规则，请转到步骤4。

3. Improve that action until we can’t anymore. Focus on this rule and find the best action for it. Draw at least 16 network specimens from the model, and then evaluate roughly 100 candidate increments to the current action, increasing geometrically in granularity as they get further from the current value. For example, evaluate r±0.01, r±0.08, r±0.64 …, taking the Cartesian product with the alternatives for m and b.

3. 不断改善这种行动，直到无法解决为止。专注于此规则并为此找到最佳action。从模型中至少产生16个网络样本，然后评估当前动作的大约100个候选增量，随着它们与当前值的距离越来越远，其粒度以几何方式增加。例如，对r±0.01，r±0.08，r±0.64…进行评估，将笛卡尔积与m和b的替代项相乘。

The modified action is evaluated by substituting it into all senders and repeating the simulation in parallel. We use the same random seed and the same set of specimen networks in the simulation of each candidate action to reduce the effects of random variation.

通过将修改后的动作替换为所有发送者，同时并行重复仿真来评估修改后的动作。我们在模拟每个候选动作时使用相同的随机种子和相同的样本网络集，以减少随机变异的影响。

If any of the candidates is an improvement, replace the action with the best new action and repeat the search, still with the same specimen networks and random seed. Otherwise, increment the epoch number of the current rule and go back to step 2.

如果有**任何**候选者有进步，使用最佳的新动作替换该动作，然后重复搜索，仍然使用相同的标本网络和随机种子。否则，增加当前规则的周期数并返回步骤2。

4. If we run out of rules in this epoch. Increment the global epoch. If the new epoch is a multiple of a parameter K, continue to step 5. Otherwise go back to step 1. We use K = 4 to balance structural improvements vs. honing the existing structure.

4.如果我们在这个时期用完规则。 增加全局周期。 如果新周期是参数K的倍数，请继续执行第5步。否则，请返回第1步。我们使用K = 4平衡结构改进与磨练现有结构。

5. Subdivide the most used rule. Recall that each rule represents a mapping from a three-dimensional rectangular region of memory space to a single action. In this step, find the most used rule, and the median memory value that triggers it. Split the rule at this point, producing eight new rules (one per dimension of the memory space), each with the same action as before. Then return to step 1.

5.细分最常用的规则。 回想一下，每个规则代表从内存空间的三维矩形区域到单个动作的映射。 在此步骤中，找到最常用的规则，以及触发它的内存中值。 此时，Remy分割规则以产生八个新规则（每个存储空间维度一个），每个规则都具有与以前相同的操作。 然后返回到步骤1。

By repeating this procedure, the structure of a RemyCC’s rule table becomes an octree [32] of memory regions. Areas of the memory space more likely to occur receive correspondingly more attention from the optimizer, and are subdivided into smaller bins that yield a more granular function relating memory to action. Which rules are more often triggered depends on every endpoint’s behaviour as well as the network’s parameters, so the task of finding the right structure for the rule table is best run alongside the process of optimizing existing rules.

通过重复此过程，RemyCC规则表的结构将成为内存区域的八叉树[32]。 内存空间中更可能出现的区域会从优化程序中获得相应的更多关注，并细分为较小的存储区，这些存储区会产生更精细的功能，将内存与动作相关联。 哪个规则触发得更多取决于每个端点的行为以及网络的参数，因此，为规则表查找正确的结构的任务最好与优化现有规则的过程一起执行。

To the best of our knowledge, this dynamic partitioning approach is novel in the context of multiagent optimization. The “greedy” approach in step 2 is key to the computational tractability and efficiency of the search because it allows us to prune the search space. Dividing the memory space into cells of different size proportional to their activity produces a rule table whose granularity is finer in regions of higher use. An improvement to consider in the future is to divide a cell only if the actions at its boundaries markedly disagree.

据我们所知，这种动态分区方法在多主体优化的背景下是新颖的。 步骤2中的“贪婪”方法是搜索的计算可处理性和效率的关键，因为它允许我们修剪搜索空间。 将内存空间按其活动成比例地划分为不同大小的单元会生成一个规则表，其规则粒度在使用率较高的区域中会更好。 将来要考虑的改进是只有在其边界处的动作明显不同时才划分一个单元

1. Other researchers have grappled with this problem; for example, Akella et al. [1] studied a restricted game, in which players are forced to obey the same particular flavour of TCP, but with the freedom to choose their additiveincrease and multiplicative decrease coefficients. Even with this constraint, the authors found that the Nash equilibrium is inefficient, unless the endpoints are restricted to run TCP Reno over a drop tail buffer, in which case the equilibrium is unfair but not inefficient running congestion control. The receiver is unchanged (as of now; this may change in the future), but is expected to send periodic ACK feedback. [↑](#footnote-ref-1)