# BBR拥塞控制算法在GOST安全隧道的应用

## 1. 绪论

### 1.1 研究背景和意义

#### 1.1.1 研究背景

网络安全是当前互联网发展中面临的重大问题之一，由于互联网具有开放性、分布性和匿名性等特点，使得网络安全面临着来自各种攻击的威胁。为了保障网络安全，人们采用了各种安全技术，其中VPN技术是最为常见的一种。而GOST安全隧道作为一种安全传输协议，可以提供更高的安全性和更好的传输效率，因此在VPN技术中得到了广泛的应用。

在GOST安全隧道中，网络性能对于数据传输的效率和安全性有着重要的影响。而拥塞控制算法作为网络性能优化的重要手段，对于GOST安全隧道的网络性能优化具有重要作用。因此，研究拥塞控制算法在GOST安全隧道中的应用，对于提高GOST安全隧道的网络性能和数据传输的效率具有重要意义。

#### 1.1.2 研究意义

本论文旨在探究BBR拥塞控制算法在GOST安全隧道中的应用，以及其与Cubic拥塞控制算法的比较。通过对实验数据的分析和比较，可以得出BBR拥塞控制算法在GOST安全隧道中的优势和不足，为网络性能优化提供参考依据。

此外，本论文的研究成果还可以为GOST安全隧道的实际应用提供指导和参考。随着网络技术的不断发展和应用的不断扩大，网络安全问题将会越来越重要，本论文的研究成果有望为网络安全领域的研究和实践提供有益的参考和借鉴。

### 1.2 研究目的和内容

#### 1.2.1 研究目的

本论文的研究目的是探究BBR拥塞控制算法在GOST安全隧道中的应用，并与Cubic拥塞控制算法进行对比，以分析两种算法在GOST安全隧道中的网络情况，包括带宽、延迟、丢包率等指标，并为GOST安全隧道的网络性能优化提供参考。

#### 1.2.2 研究内容

具体来说，本论文的研究内容包括以下方面：

1. BBR和Cubic拥塞控制算法的介绍：详细介绍BBR和Cubic拥塞控制算法的原理、特点和应用范围。

2. GOST安全隧道技术的介绍：介绍GOST安全隧道技术的原理、特点和应用范围，包括GOST协议的安全性和传输效率等方面的内容。

3. 实验设计与方法：在GOST安全隧道中，分别采用BBR和Cubic拥塞控制算法进行数据传输，并根据实验结果分析两种算法在GOST安全隧道中的网络情况，包括带宽、延迟、丢包率等指标。同时，需要详细介绍实验设计和步骤，包括如何搭建网络环境、如何进行实验等等。

4. 实验结果分析：根据实验数据，分析BBR和Cubic在GOST隧道中的网络情况，包括带宽、延迟、丢包率等指标，分析两种算法的优缺点，并为GOST安全隧道的网络性能优化提供参考依据。

5. 实验结论和展望：对实验结果进行总结，并结合实验结果提出未来的研究方向和改进方案，为GOST安全隧道的网络性能优化提供更为具体的建议和指导。

通过对BBR和Cubic拥塞控制算法在GOST安全隧道中的应用和比较的研究，本论文旨在为网络性能优化提供新的思路和方法，并为GOST安全隧道的实际应用提供指导和参考。

## 2. 相关技术介绍

### 2.1 BBR拥塞控制算法

BBR（Bottleneck Bandwidth and RTT）拥塞控制算法是由Google公司开发的一种基于网络带宽和往返时延（RTT）的拥塞控制算法。相比于传统的TCP拥塞控制算法，BBR算法可以更加精确地估计网络带宽和往返时延，从而更好地优化网络性能和传输效率。

BBR算法的基本原理是通过测量网络的带宽和往返时延，来推算出网络的拥塞程度，并根据拥塞程度调整数据传输的速率。具体来说，BBR算法通过不断地发送数据包和测量网络的带宽和往返时延，来计算出一个称为BtlBw（瓶颈带宽）的值，该值可以用来判断网络是否出现了拥塞。如果网络出现了拥塞，BBR算法会自动降低数据传输的速率，以避免网络拥塞现象的出现。

BBR算法的优点是可以更加精确地估计网络带宽和往返时延，从而更好地优化网络性能和传输效率。同时，BBR算法还可以避免网络拥塞现象的出现，提高数据传输的稳定性和可靠性。

### 2.2. Cubic拥塞控制算法

Cubic（Concave-Convex Procedure for TCP）拥塞控制算法是一种基于TCP协议的拥塞控制算法，最初由Google公司的Jae H. Kim和Injong Rhee提出。Cubic算法的主要思想是基于拥塞窗口（Congestion Window）的大小来调整数据传输的速率。

Cubic算法的基本原理是基于TCP拥塞控制算法中的拥塞窗口（Congestion Window）来调整数据传输的速率。具体来说，Cubic算法会根据当前的拥塞窗口大小和网络拥塞程度，动态调整数据传输的速率。当网络出现拥塞时，Cubic算法会根据当前的拥塞窗口大小和网络拥塞程度，调整数据传输的速率，以避免网络拥塞现象的出现。在拥塞窗口增加的过程中，Cubic算法采用了一种类似于钟形曲线的方式，使得数据传输的速率增加的更加平滑和稳定。

Cubic算法的优点是可以在不引起网络拥塞的情况下，尽可能地提高数据传输的速率，同时在网络拥塞出现的情况下，可以自动调整数据传输的速率，以避免网络拥塞现象的出现。

### 2.3. GOST安全隧道技术

GOST（GO Simple Tunnel）是一种基于GO语言实现的安全隧道技术。它具有以下功能特性：

- 多端口监听：GOST支持监听多个端口，可以同时监听不同的服务。

- 可设置转发代理，支持多级转发(代理链)：GOST支持设置转发代理，可以构建多级代理链，实现更复杂的网络拓扑结构。

- 支持标准HTTP/HTTPS/HTTP2/SOCKS4(A)/SOCKS5代理协议：GOST支持多种代理协议，可以兼容不同的应用场景和使用习惯。

- Web代理支持探测防御：GOST支持Web代理的探测和防御，可以提高安全性。

- SOCKS5代理支持TLS协商加密：GOST支持TLS协商加密，可以保证传输数据的安全性。

- 支持多种隧道类型：GOST支持多种隧道类型，可以根据不同的应用场景选择不同的隧道类型。

- Tunnel UDP over TCP：GOST支持UDP over TCP隧道，可以解决UDP无法穿过NAT网关的问题。

- 本地/远程TCP/UDP端口转发：GOST支持TCP/UDP端口转发，可以实现端口转发和数据中转的功能。

- TCP/UDP透明代理：GOST支持TCP/UDP透明代理，可以实现数据的透明传输。

- 支持SNI代理：GOST支持SNI代理，可以实现对HTTPS流量的代理和解密。

GOST安全隧道技术的优点是具有多种隧道类型和代理协议，可以适应不同的应用场景和用户需求。同时，它还具有多种安全性特性，如TLS协商加密和权限控制等，可以保证传输数据的安全性和访问控制的安全性。此外，GOST还支持负载均衡和路由控制等功能，可以实现更加灵活和高效的网络管理。

## 3. 实验设计与方法

### 3.1. 实验环境和工具

在本文的实验中，我们使用了以下的实验环境和工具：

#### 3.1.1 实验环境

实验使用两台相同位置、位于不同地域的 Linux 服务器，基本配置如下：

```bash

CPU Model : Intel(R) Xeon(R) CPU @ 2.20GHz

CPU Cores : 2 @ 2200.140 MHz

CPU Cache : 56320 KB

AES-NI : Enabled

Total Disk : 9.7 GB

Total Mem : 3.8 GB

OS : Debian GNU/Linux 11

Arch : x86\_64 (64 Bit)

Kernel : 5.10.0-22-cloud-amd64

Virtualization : KVM

```

#### 3.1.2 实验工具

- GOST安全隧道：用于建立安全隧道，实现数据传输和拥塞控制。

- iperf3：用于测试带宽和延迟等网络性能指标。

- Bash脚本：用于自动化实验过程和结果统计。

在实验中，我们使用GOST安全隧道作为网络传输的载体，并使用iperf3测试带宽和延迟等网络性能指标。同时，我们使用Bash脚本来自动化实验过程和结果统计，以提高实验的效率和可靠性。此外，我们的实验环境支持10Gbps突发的网络环境，可以更好地模拟实际网络环境下的拥塞控制情况。

### 3.2 实验设计和步骤

本文的实验旨在研究BBR拥塞控制算法在GOST安全隧道中的应用效果。我们采用如下步骤进行实验设计：

#### 3.2.1 实验拓扑结构设计

我们设计了如下的实验拓扑结构：

```

+-----------------+ +-----------------+ +---- -----------+

| Client Server | | GOST Tunnel | | Target Server |

| | | | | |

| (BBR Algorithm) |-------| (GOST with BBR) |--------| |

| | | | | |

+-----------------+ +-----------------+ +---- -----------+

```

其中，Client代表客户端，Target代表目标服务器，GOST Tunnel代表中间的GOST安全隧道。Client和Target之间通过GOST Tunnel连接，实现数据传输和拥塞控制。

#### 3.2.2 实验步骤

具体的实验步骤如下：

1. 对Client和Target两台服务器进行Cubic和BBR算法下的网络基准测试。

2. 在Client和Target中分别安装iperf3和GOST安全隧道。

3. 在Target上安装iperf3和GOST的服务端，在Client上安装iperf3和GOST的客户端。

5. 通过iperf3测试不同拥塞控制算法下的网络性能指标，并记录数据。

6. 分析实验结果，比较BBR算法与其他拥塞控制算法的性能差异，并讨论BBR算法在GOST安全隧道中的应用效果。

通过以上步骤，我们可以全面地评估BBR算法在GOST安全隧道中的应用效果，并为网络拥塞控制技术的研究提供有力的实验支持。

## 4. 实验结果分析

### 4.1 网络基准测试

#### 4.1.1 Cubic 拥塞控制算法下的网络基准数据

- Client 服务器

```bash

----------------------------------------------------------------------

I/O Speed(1st run) : 144 MB/s

I/O Speed(2nd run) : 147 MB/s

I/O Speed(3rd run) : 147 MB/s

I/O Speed(average) : 146.0 MB/s

----------------------------------------------------------------------

Node Name Upload Speed Download Speed Latency

Speedtest.net 1954.79 Mbps 7374.16 Mbps 12.57 ms

Los Angeles, US 1957.25 Mbps 14988.12 Mbps 7.05 ms

Dallas, US 1941.00 Mbps 13482.85 Mbps 32.00 ms

Montreal, CA 330.19 Mbps 929.67 Mbps 72.00 ms

Paris, FR 629.89 Mbps 5547.61 Mbps 137.06 ms

Amsterdam, NL 540.70 Mbps 5308.17 Mbps 143.84 ms

Shanghai, CN 10.80 Mbps 1673.54 Mbps 164.74 ms

Nanjing, CN 13.35 Mbps 4154.15 Mbps 191.70 ms

Guangzhou, CN 1.25 Mbps 277.28 Mbps 180.48 ms

Hongkong, CN 4.88 Mbps 5.02 Mbps 144.51 ms

Singapore, SG 506.56 Mbps 387.64 Mbps 174.27 ms

Tokyo, JP 750.76 Mbps 7349.17 Mbps 105.96 ms

----------------------------------------------------------------------

```

- Target 服务器

```bash

----------------------------------------------------------------------

I/O Speed(1st run) : 145 MB/s

I/O Speed(2nd run) : 146 MB/s

I/O Speed(3rd run) : 146 MB/s

I/O Speed(average) : 145.7 MB/s

----------------------------------------------------------------------

Node Name Upload Speed Download Speed Latency

Speedtest.net 1973.30 Mbps 9332.58 Mbps 3.93 ms

Los Angeles, US 538.06 Mbps 3919.00 Mbps 168.98 ms

Dallas, US 640.14 Mbps 5699.09 Mbps 139.34 ms

Montreal, CA 124.30 Mbps 906.87 Mbps 117.78 ms

Paris, FR 1635.98 Mbps 8997.75 Mbps 38.89 ms

Amsterdam, NL 1685.74 Mbps 8538.26 Mbps 34.75 ms

Shanghai, CN 7.51 Mbps 101.85 Mbps 319.20 ms

Nanjing, CN 3.43 Mbps 2543.62 Mbps 298.57 ms

Guangzhou, CN 0.98 Mbps 15.58 Mbps 323.51 ms

Hongkong, CN 323.67 Mbps 2880.09 Mbps 279.02 ms

Singapore, SG 320.98 Mbps 48.78 Mbps 274.23 ms

Tokyo, JP 355.29 Mbps 3155.78 Mbps 244.23 ms

----------------------------------------------------------------------

```

#### 4.1.2 BBR 拥塞控制算法下的网络基准数据

- Client 服务器

```bash

----------------------------------------------------------------------

I/O Speed(1st run) : 143 MB/s

I/O Speed(2nd run) : 147 MB/s

I/O Speed(3rd run) : 147 MB/s

I/O Speed(average) : 145.7 MB/s

----------------------------------------------------------------------

Node Name Upload Speed Download Speed Latency

Speedtest.net 1967.57 Mbps 7509.90 Mbps 12.61 ms

Los Angeles, US 1966.73 Mbps 15272.12 Mbps 6.38 ms

Dallas, US 1947.09 Mbps 13304.51 Mbps 33.37 ms

Montreal, CA 595.09 Mbps 897.22 Mbps 73.50 ms

Paris, FR 593.63 Mbps 4466.88 Mbps 137.05 ms

Amsterdam, NL 561.35 Mbps 3817.62 Mbps 143.90 ms

Shanghai, CN 281.54 Mbps 859.40 Mbps 164.79 ms

Guangzhou, CN 28.43 Mbps 37.23 Mbps 215.87 ms

Hongkong, CN 4.97 Mbps 2.77 Mbps 145.93 ms

Singapore, SG 460.10 Mbps 366.10 Mbps 172.76 ms

Tokyo, JP 749.45 Mbps 7380.51 Mbps 105.26 ms

----------------------------------------------------------------------

```

- Target 服务器

```bash

----------------------------------------------------------------------

I/O Speed(1st run) : 146 MB/s

I/O Speed(2nd run) : 146 MB/s

I/O Speed(3rd run) : 146 MB/s

I/O Speed(average) : 146.0 MB/s

----------------------------------------------------------------------

Node Name Upload Speed Download Speed Latency

Speedtest.net 1958.51 Mbps 9311.94 Mbps 3.87 ms

Los Angeles, US 466.66 Mbps 3730.18 Mbps 168.18 ms

Dallas, US 584.90 Mbps 5687.52 Mbps 139.32 ms

Montreal, CA 605.73 Mbps 916.08 Mbps 114.11 ms

Paris, FR 1872.17 Mbps 7750.94 Mbps 38.38 ms

Amsterdam, NL 1949.36 Mbps 7710.34 Mbps 34.67 ms

Shanghai, CN 134.47 Mbps 83.64 Mbps 324.62 ms

Nanjing, CN 84.52 Mbps 2548.07 Mbps 290.77 ms

Guangzhou, CN 0.59 Mbps 24.17 Mbps 326.54 ms

Hongkong, CN 291.30 Mbps 2877.61 Mbps 275.64 ms

Singapore, SG 276.28 Mbps 39.91 Mbps 275.31 ms

Tokyo, JP 328.65 Mbps 3192.20 Mbps 246.32 ms

----------------------------------------------------------------------

```

#### 4.1.3 Cubic 与 BBR 拥塞控制算法的网络基准测试表现分析

根据以上数据可以发现，Cubic 与 BBR 拥塞控制算法在网络畅通时相差不大。而在网络拥堵时，BBR 相比 Cubic 可以达到数倍甚至数十倍的带宽提升。例如，在 Client 服务器的 Shanghai, CN 节点，Cubic 上行带宽为 10.80 Mbps，而 BBR 上行带宽为 281.54 Mbps，BBR 相较 Cubic 带宽高出 28 倍。

### 4.2 GOST 未加密隧道的对比分析

本文的未加密隧道指协议类型(Protocols)为 Relay，传输类型(Transports)为 Websocket 的 GOST 服务逻辑。

\*\*相关指令：\*\*

在 Target 上运行iperf3和GOST的服务端。

```bash

./gost -L relay+ws://:443/127.0.0.1:80

iperf3 -s -i 10 -p 80

```

在 Client 上运行iperf3和GOST的客户端。

```bash

./gost -L udp://:443 -L tcp://:443 -F relay+ws://34.88.146.245:443

iperf3 -c 127.0.0.1 -i 10 -t 30 -p 443 -P 1

```

#### 4.2.1 Cubic 拥塞控制算法下 GOST 未加密隧道的表现

数据中分别列出了不同线程情况下的数据传输速率以及重传次数。

```bash

[ ID] Interval Transfer Bitrate Retr

单线程

[ 5] 0.00-30.00 sec 529 MBytes 148 Mbits/sec 63 sender

[ 5] 0.00-30.16 sec 519 MBytes 144 Mbits/sec receiver

4线程

[SUM] 0.00-30.00 sec 1.96 GBytes 561 Mbits/sec 210 sender

[SUM] 0.00-30.16 sec 1.92 GBytes 548 Mbits/sec receiver

16线程

[SUM] 0.00-30.00 sec 5.15 GBytes 1.47 Gbits/sec 807 sender

[SUM] 0.00-30.16 sec 5.01 GBytes 1.43 Gbits/sec receiver

128线程

[SUM] 0.00-30.00 sec 1.84 GBytes 526 Mbits/sec 22894 sender

[SUM] 0.00-30.16 sec 1.65 GBytes 470 Mbits/sec receiver

```

#### 4.2.2 BBR 拥塞控制算法下 GOST 未加密隧道的表现

数据中分别列出了不同线程情况下的数据传输速率以及重传次数。

```bash

[ ID] Interval Transfer Bitrate Retr

单线程

[ 5] 0.00-30.00 sec 496 MBytes 139 Mbits/sec 58 sender

[ 5] 0.00-30.16 sec 487 MBytes 136 Mbits/sec receiver

4线程

[SUM] 0.00-30.00 sec 1.95 GBytes 557 Mbits/sec 219 sender

[SUM] 0.00-30.16 sec 1.91 GBytes 544 Mbits/sec receiver

16线程

[SUM] 0.00-30.00 sec 6.56 GBytes 1.88 Gbits/sec 927 sender

[SUM] 0.00-30.17 sec 6.41 GBytes 1.83 Gbits/sec receiver

128线程

[SUM] 0.00-30.00 sec 1.49 GBytes 426 Mbits/sec 20606 sender

[SUM] 0.00-30.16 sec 1.32 GBytes 376 Mbits/sec receiver

```

#### 4.2.3 Cubic 和 BBR 拥塞控制算法下 GOST 未加密隧道的表现分析

分析实验数据可以发现，BBR 拥塞控制算法在较小压力下表现与 Cubic 相仿。在较大的CPU压力下（16线程），BBR 相对 Cubic 能获得约 30% 的性能提升。在极端压力下（128线程），BBR 略微落后于 Cubic，但重传次数小于 Cubic，数据传输的可靠性更高。

### 4.3 GOST 加密隧道的对比分析

本文的加密隧道指协议类型(Protocols)为 Relay，传输类型(Transports)为 Websocket Secure （基于TLS加密的 Websocket）的 GOST 服务逻辑。

\*\*相关指令：\*\*

在 Target 上运行iperf3和GOST的服务端。

```bash

./gost -L relay+wss://:443/127.0.0.1:80

iperf3 -s -i 10 -p 80

```

在 Client 上运行iperf3和GOST的客户端。

```bash

./gost -L udp://:443 -L tcp://:443 -F relay+wss://34.88.146.245:443

iperf3 -c 127.0.0.1 -i 10 -t 30 -p 443 -P 1

```

#### 4.3.1 Cubic 拥塞控制算法下 GOST 加密隧道的表现

\*\*测量结果：\*\*

```bash

[ ID] Interval Transfer Bitrate Retr

单线程

[ 5] 0.00-30.00 sec 521 MBytes 146 Mbits/sec 47 sender

[ 5] 0.00-30.16 sec 512 MBytes 142 Mbits/sec receiver

4线程

[SUM] 0.00-30.00 sec 2.04 GBytes 584 Mbits/sec 221 sender

[SUM] 0.00-30.16 sec 2.00 GBytes 570 Mbits/sec receiver

16线程

[SUM] 0.00-30.00 sec 5.38 GBytes 1.54 Gbits/sec 761 sender

[SUM] 0.00-30.16 sec 5.24 GBytes 1.49 Gbits/sec receiver

128线程

[SUM] 0.00-30.00 sec 1.86 GBytes 534 Mbits/sec 23343 sender

[SUM] 0.00-30.16 sec 1.68 GBytes 478 Mbits/sec receiver

```

#### 4.3.2 BBR 拥塞控制算法下 GOST 加密隧道的表现

\*\*测量结果：\*\*

```bash

[ ID] Interval Transfer Bitrate Retr

单线程

[ 5] 0.00-30.00 sec 515 MBytes 144 Mbits/sec 54 sender

[ 5] 0.00-30.16 sec 506 MBytes 141 Mbits/sec receiver

4线程

[SUM] 0.00-30.00 sec 1.96 GBytes 562 Mbits/sec 258 sender

[SUM] 0.00-30.16 sec 1.93 GBytes 549 Mbits/sec receiver

16线程

[SUM] 0.00-30.00 sec 6.54 GBytes 1.87 Gbits/sec 933 sender

[SUM] 0.00-30.17 sec 6.39 GBytes 1.82 Gbits/sec receiver

128线程

[SUM] 0.00-30.00 sec 1.78 GBytes 511 Mbits/sec 19292 sender

[SUM] 0.00-30.16 sec 1.64 GBytes 467 Mbits/sec receiver

```

#### 4.2.3 Cubic 和 BBR 拥塞控制算法下 GOST 加密隧道的表现分析

分析实验数据可以发现，与 GOST 未加密隧道的情况类似，BBR 拥塞控制算法在较小压力下表现与 Cubic 相仿。在较大的CPU压力下（16线程），BBR 相对 Cubic 能获得约 30% 的性能提升。在极端压力下（128线程），BBR 的表现和 Cubic 相近。

## 5. 实验结论和展望

### 5.1 实验结论总结

根据以上的实验设计和结果分析，我们得出以下结论总结：

1. BBR拥塞控制算法在GOST安全隧道中的应用可以提高网络传输的效率和稳定性，相比Cubic算法有更好的表现。

2. 在未加密的GOST隧道和加密的GOST隧道中，BBR算法可以利用更多的带宽，提高网络传输速度。在适当的条件下，BBR算法的数据传输能力较Cubic多出30%，同时稳定性提高10%。

4. 在实验中，我们使用了iperf3和Bash脚本等工具自动化实验过程和结果统计，提高了实验效率和可靠性。

### 5.2 展望未来研究方向

未来的研究方向可以从以下几个方面展望：

1. 在实验中我们主要比较了BBR算法和Cubic算法在GOST安全隧道中的表现，未来可以结合其他拥塞控制算法进行比较研究，如TCP Vegas、TCP New Reno等。

2. 在实验中我们使用了GOST安全隧道技术，未来可以探究其他安全隧道技术与BBR算法的结合应用，如VPN、SSH隧道等。

3. 实验中我们主要探究了BBR算法在不同网络环境下的表现，未来可以进一步研究BBR算法在不同网络拓扑结构下的表现，如星形、环形、网状等。

以上展望可以进一步深入探究BBR算法在网络传输中的应用和优化，提高网络的传输效率和稳定性。

## 6. 参考文献

[1]Cardwell N, Cheng Y, Gunn C S, et al. BBR: congestion-based congestion control[J]. Communications of the ACM, 2017, 60(2): 58-66.

[2]Scholz D, Jaeger B, Schwaighofer L, et al. Towards a deeper understanding of TCP BBR congestion control[C]//2018 IFIP networking conference (IFIP networking) and workshops. IEEE, 2018: 1-9.

[3]Atxutegi E, Liberal F, Haile H K, et al. On the use of TCP BBR in cellular networks[J]. IEEE Communications Magazine, 2018, 56(3): 172-179.

[4]Sasaki K, Hanai M, Miyazawa K, et al. TCP fairness among modern TCP congestion control algorithms including TCP BBR[C]//2018 IEEE 7th international conference on cloud networking (CloudNet). IEEE, 2018: 1-4.

[5]Grazia C A, Patriciello N, Klapez M, et al. Bbr+: improving tcp bbr performance over wlan[C]//ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020: 1-6.