第六讲
协议算法
Lecture 6
Protocol Algorithm

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

协议算法
Protocol Algorithms

- 计算机网络在现代生活中的作用毋庸置疑；在整个架构设计与功能结构中，多种算法配合相关网络硬件，综合作用以保证相关功能在各个层面的运作。其一大类算法具有典型性，了解这类算法的思想对解决相关问题有积极的启发意义。

  It is undoubted that the networking plays an important role in daily life. In the overall architectural design and functional structuring, various of algorithms coupled with networking hardware, operate together to insure the performance on different levels. A large category of algorithms are stereotypical, and to understand the methodologies of these algorithms can inspire problem-solving of similar kinds.

协议算法
Protocol Algorithms

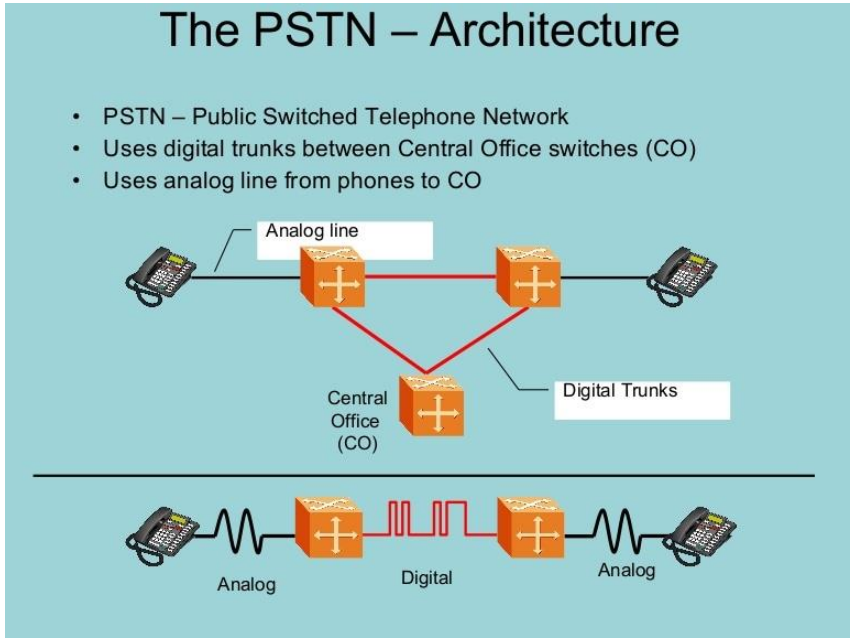
- 从大算法的角度，计算机网络方便的技术实施逐渐趋于标准化。一种指导性方式是分层思想，即通过抽象将功能分层，各层仅负责本层功能实现；另一种是将控制与数据分离，即将数据传输与调控数据传输分别处理，分别称为数据层面与控制层面。

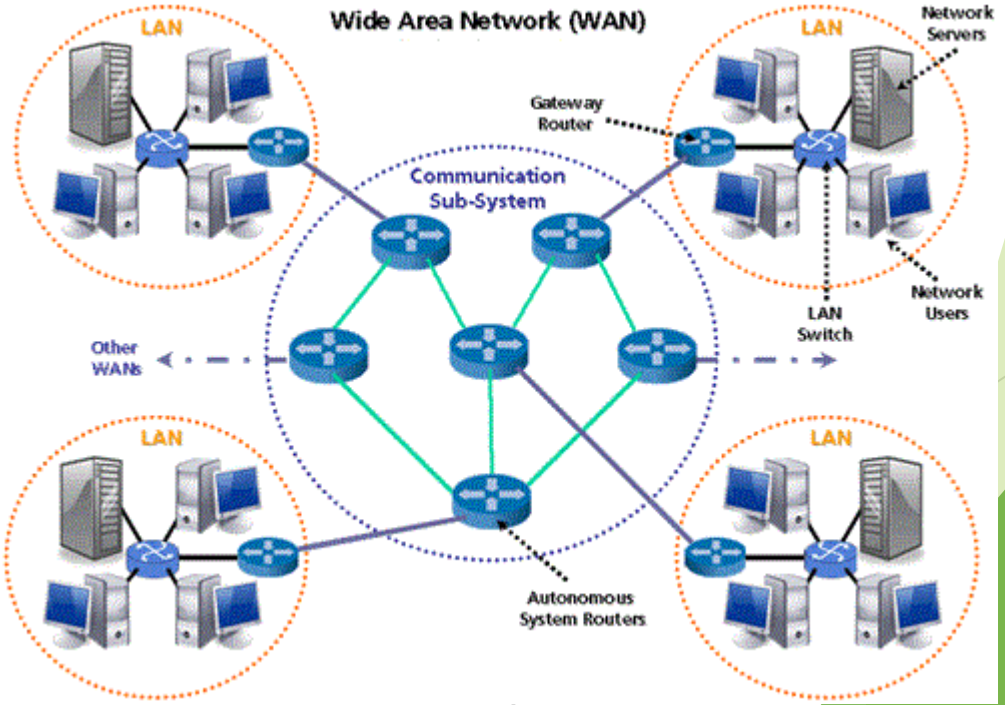
  From a broader perspective of algorithm, the practice in computer networking is more and more standardized. The first guidance is the layer-wise modelling by abstraction. Each layer is only responsible for work targeted at itself, and these works are transparency to other layers. The second philosophy is detachment of controlling from data, aka, separate the data transmission from the controlling mechanism of data transmission, called data plane and control plane, respectively.

协议算法
Protocol Algorithms

- 在讲相关算法之前，我们先回顾一下公共交换电话网与计算机网络之间差别。

  Before we proceed to some algorithms, let's have a quick review of the public switched telephone network and computer network.

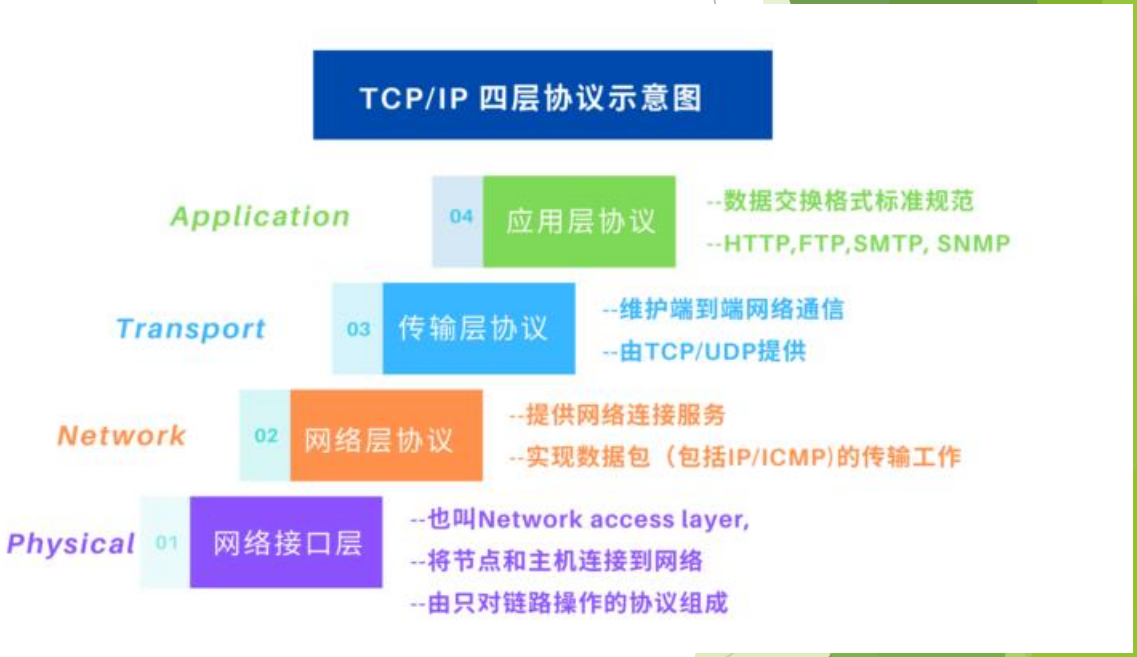




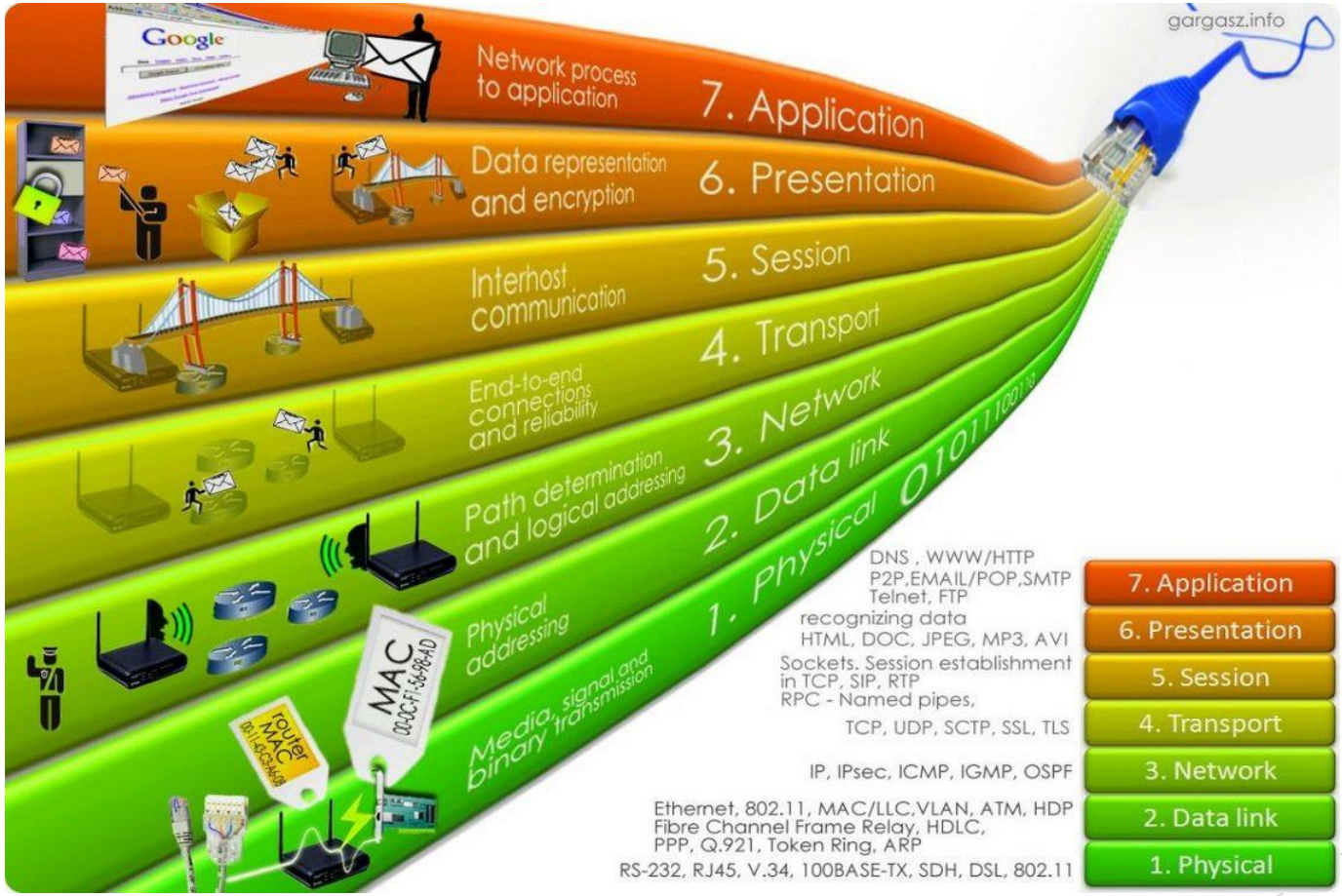
协议算法
Protocol Algorithms

- 计算机网络中最大特点是分布式的，其节点在相互通讯之前并不需要预先建立连接，数据包在节点传送时，也是尽力而为；最重要的协议簇，即TCP/IP协议集。

  The most eminent characteristic of computer network is the distributed property, and there is no necessary to establish the connection prior to communication. The nodes offer a best-effort service of delivering datagrams between hosts. And the most significant protocol suite in the computer networking field is the TCP/IP protocols.



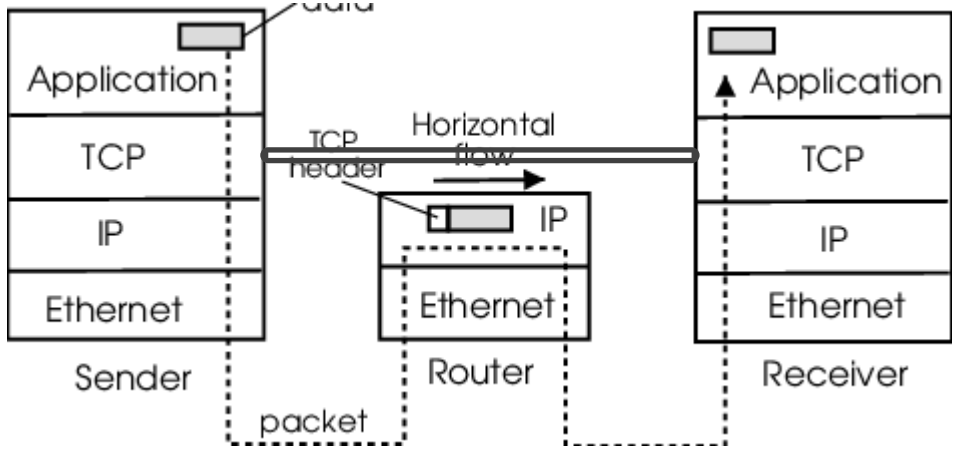
协议算法
Protocol Algorithms



协议算法
Protocol Algorithms

- 在TCP/IP协议集中，仿传统PSTN的面向流的TCP连接，是偏向逻辑意义上的，这点从路由器的视角看，更为清晰。

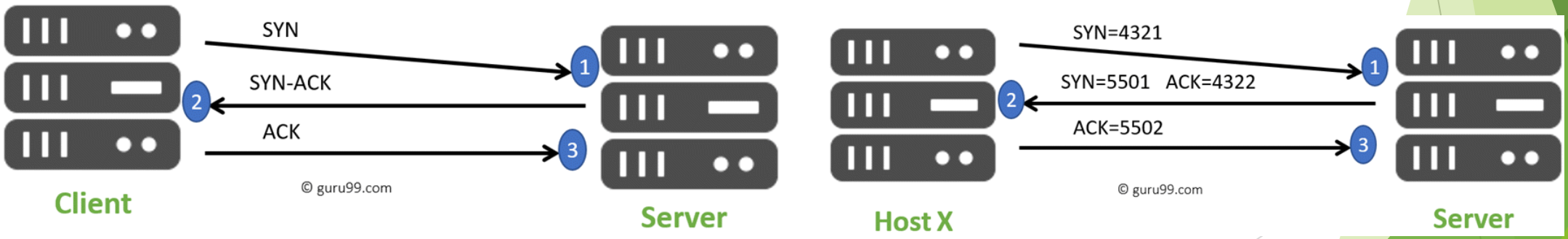
  In the TCP/IP protocol suite, the stream-oriented TCP connections analogous to the concepts in traditional PSTN are from the logical perspective, this is clearer from the router's perspective.



协议算法
Protocol Algorithms

- 面向流的TCP连接强调在不可靠的传输机制上，建立一个可靠的连接或流式传输。TCP的三次握手协议，或三次握手算法，用以保证一个连接的建立。

  TCP requires a way to build the stream-oriented communication between hosts upon an un-reliable transmission mechanism. The three-way handshake algorithm of TCP is to make sure the successful establishment of a TCP connection.



协议算法
Protocol Algorithms

- 我们以回环接口上的客户端与服务器模型来查看TCP建立连接的三次握手过程，注意发起连接的一方的SYN位与确认连接的另一方的ACK位的变化。

  We demonstrate the three-way handshake of TCP connection initiation by implementing a client/server model over the loop interface. Note the values of SYN and ACK on each peer during the process.

	Time	Source	Destination	Protoc	Lengt	Info
87	24.634582	127.0.0.1	127.0.0.1	TCP	56	60060 → 27015 [SYN] Seq=3373982931 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
88	24.634653	127.0.0.1	127.0.0.1	TCP	56	27015 → 60060 [SYN, ACK] Seq=1679585359 Ack=3373982932 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
89	24.634719	127.0.0.1	127.0.0.1	TCP	44	60060 → 27015 [ACK] Seq=3373982932 Ack=1679585360 Win=2619648 Len=0
90	24.634776	127.0.0.1	127.0.0.1	TCP	58	60060 → 27015 [PSH, ACK] Seq=3373982932 Ack=1679585360 Win=2619648 Len=14
91	24.634805	127.0.0.1	127.0.0.1	TCP	44	27015 → 60060 [ACK] Seq=1679585360 Ack=3373982946 Win=2619648 Len=0
92	24.699800	127.0.0.1	127.0.0.1	TCP	44	60060 → 27015 [FIN, ACK] Seq=3373982946 Ack=1679585360 Win=2619648 Len=0
93	24.699879	127.0.0.1	127.0.0.1	TCP	44	27015 → 60060 [ACK] Seq=1679585360 Ack=3373982947 Win=2619648 Len=0
94	24.700011	127.0.0.1	127.0.0.1	TCP	58	27015 → 60060 [PSH, ACK] Seq=1679585360 Ack=3373982947 Win=2619648 Len=14
95	24.700061	127.0.0.1	127.0.0.1	TCP	44	60060 → 27015 [ACK] Seq=3373982947 Ack=1679585374 Win=2619648 Len=0
96	24.700873	127.0.0.1	127.0.0.1	TCP	44	27015 → 60060 [FIN, ACK] Seq=1679585374 Ack=3373982947 Win=2619648 Len=0
97	24.700965	127.0.0.1	127.0.0.1	TCP	44	60060 → 27015 [ACK] Seq=3373982947 Ack=1679585375 Win=2619648 Len=0

协议算法
Protocol Algorithms

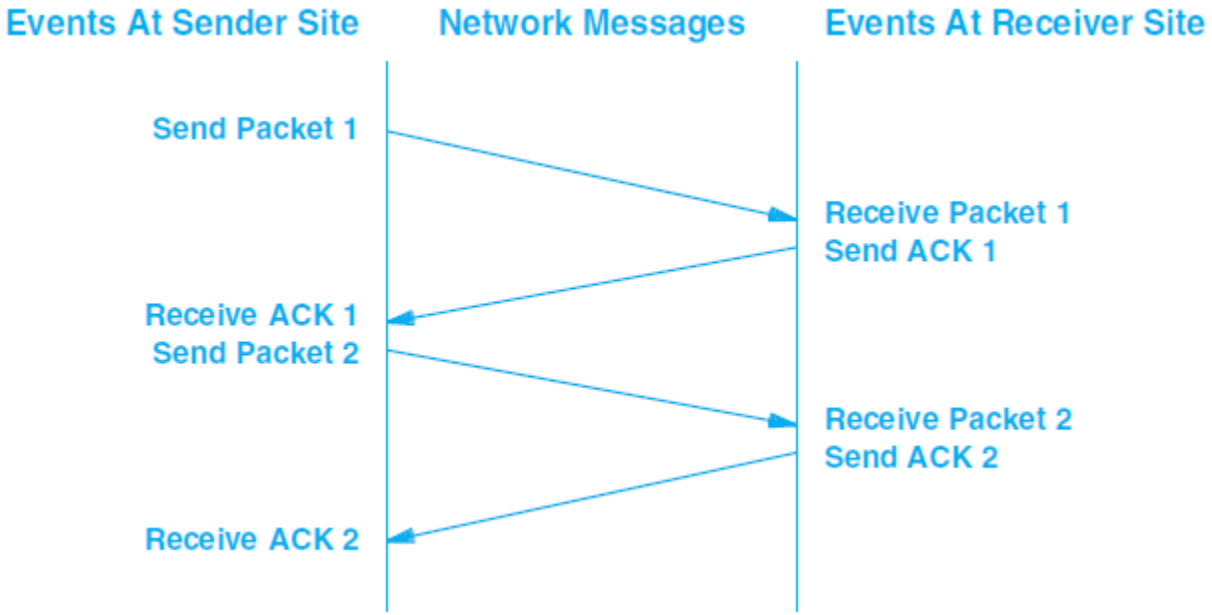
- 当连接建立起来之后，TCP数据包仍然由非可靠的IP包承载传播，依然需要依靠适当的算法，保证逻辑上的面向连接的传播特性。

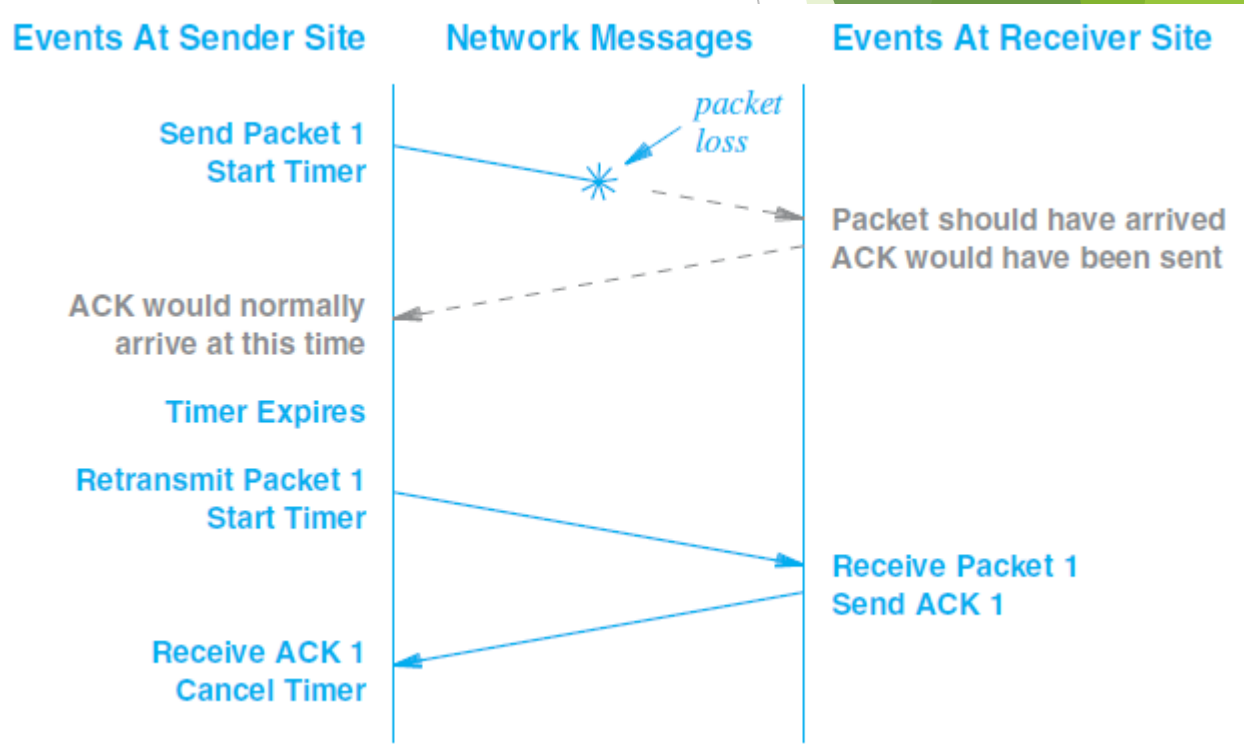
  After the establishment of the connection, the TCP packets are encapsulated into the unreliable IP packets for transmitting. It relies on certain algorithms to ensure the connection-oriented properties.

- TCP采用一种确认重传机制，来保证面向连接的服务的可靠性。

  TCP implement an acknowledge and retransmit algorithm to ensure the reliability of some connection-oriented services built upon TCP.

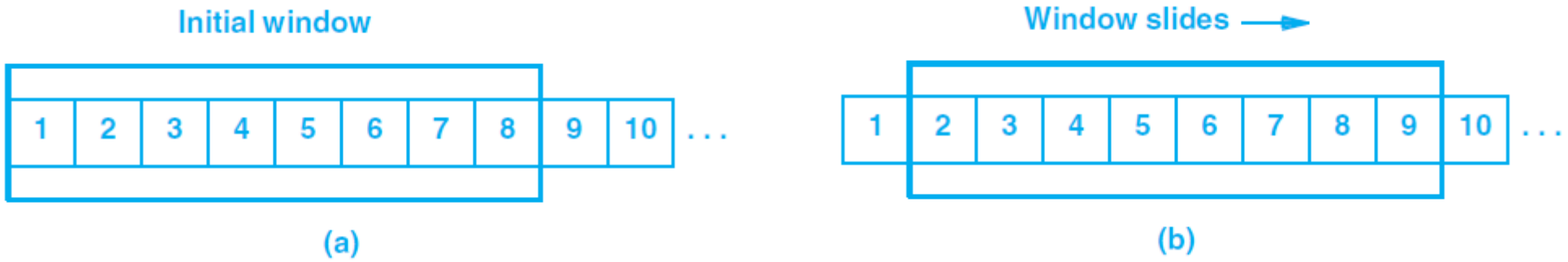
协议算法
Protocol Algorithms





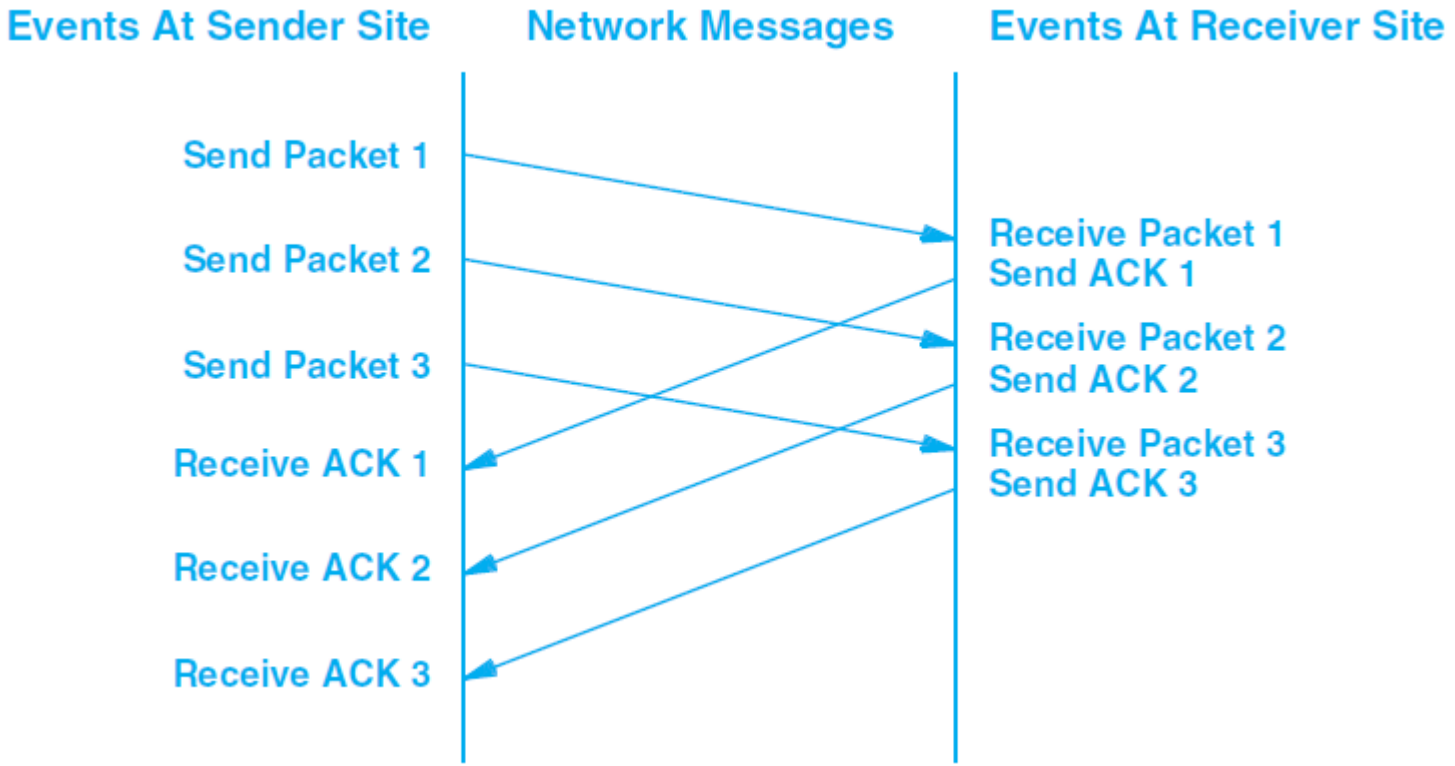
协议算法
Protocol Algorithms

- 确认重传机制需要合适的算法来实现，TCP利用滑动窗口机制保证面向流的传输的逻辑完整性：
  1. 确定每次传输的数据包的大小；
  2. 初始化一个包含若干数据包的窗口；
  3. 依次发送数据包，等待对端确认；
  4. 如果最初发送的数据包得到了确认，则将窗口向右滑动；
  5. 如果某个数据包长时间得不到确认，则重传。



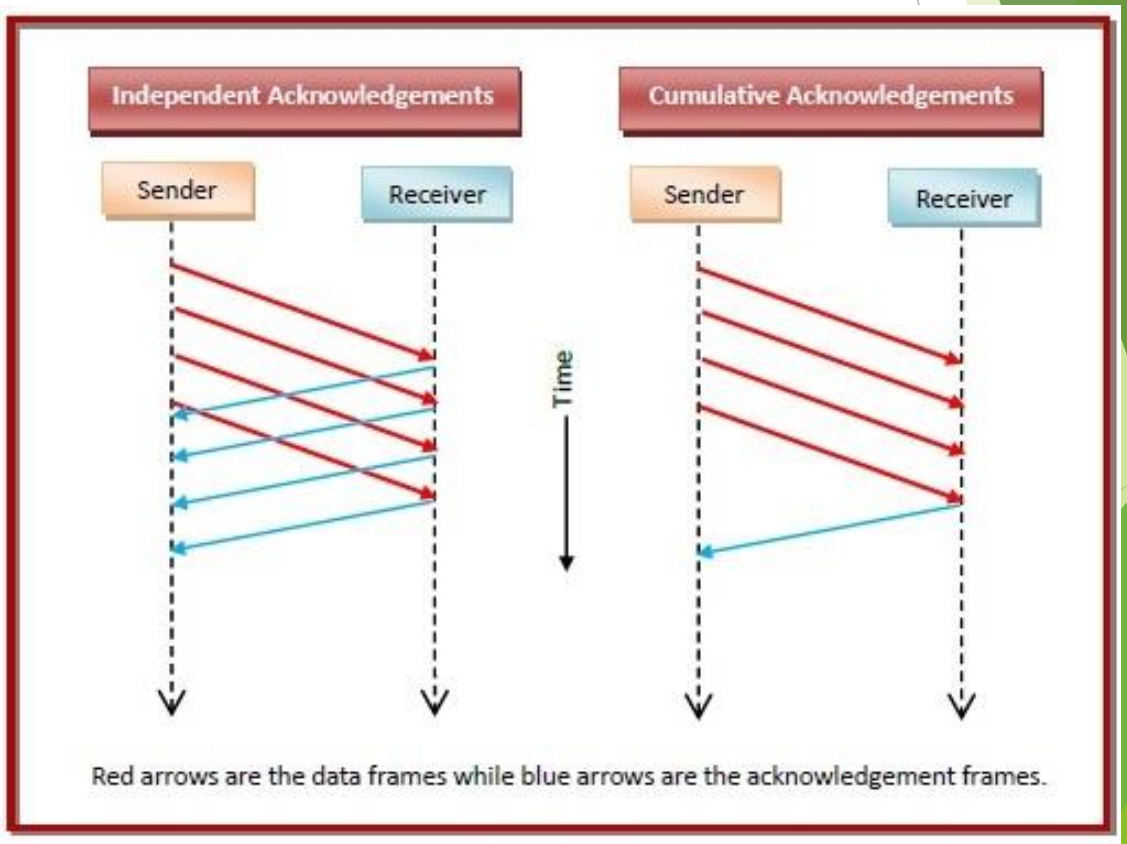
协议算法
Protocol Algorithms

- 利用滑动窗口机制可以保证网络在一定程度的满载，从而提高网络利用率；通过合适的机制，如累积确认，或进一步的，选择确认，则同时可以提高端点的通讯效率：



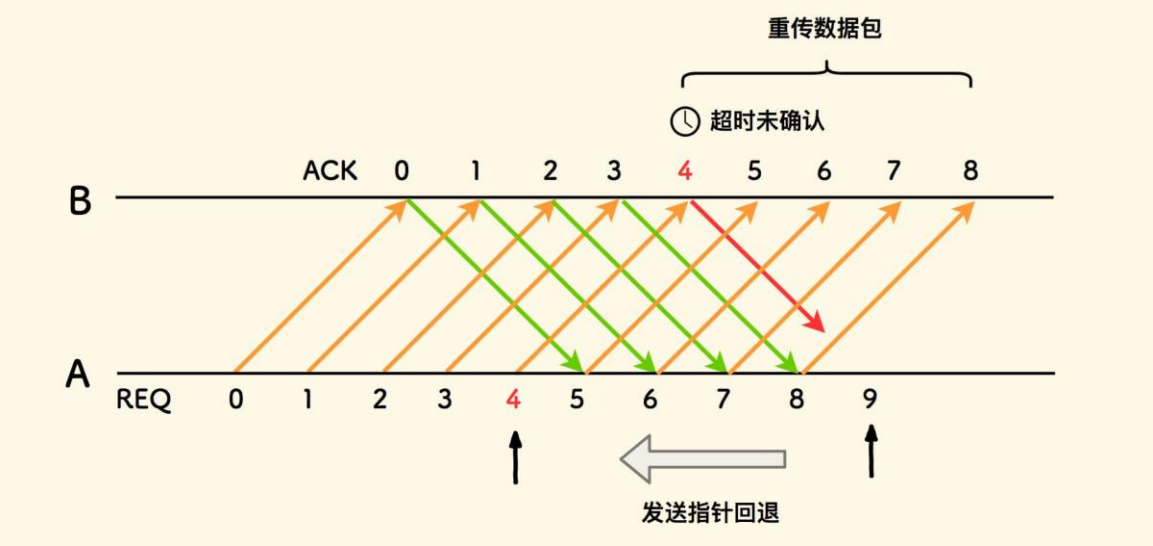
协议算法
Protocol Algorithms

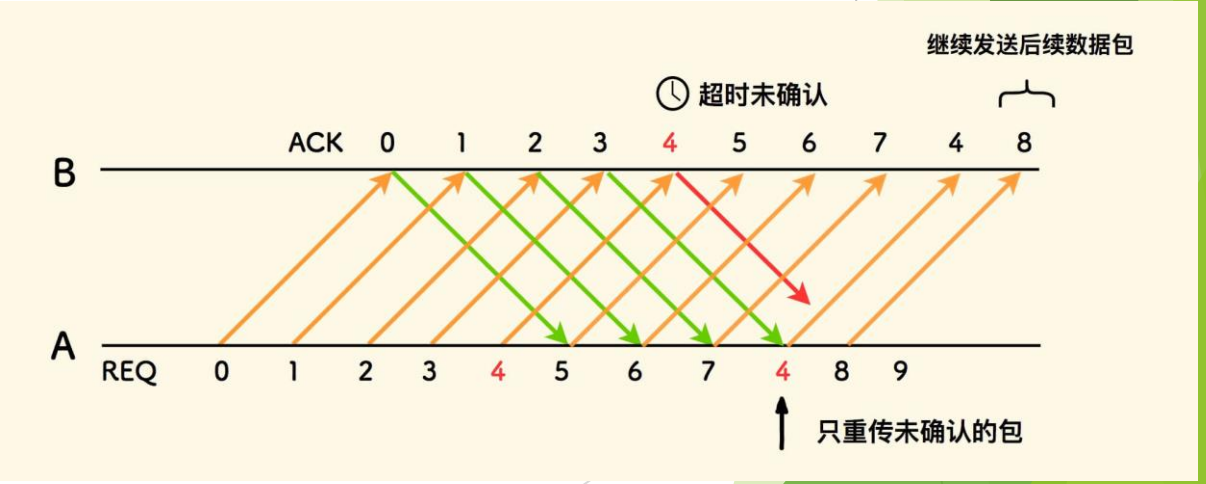
- 利用滑动窗口机制可以保证网络在一定程度的满载，从而提高网络利用率；通过合适的机制，如累积确认，或进一步的，选择确认，则同时可以提高端点的通讯效率：



协议算法
Protocol Algorithms

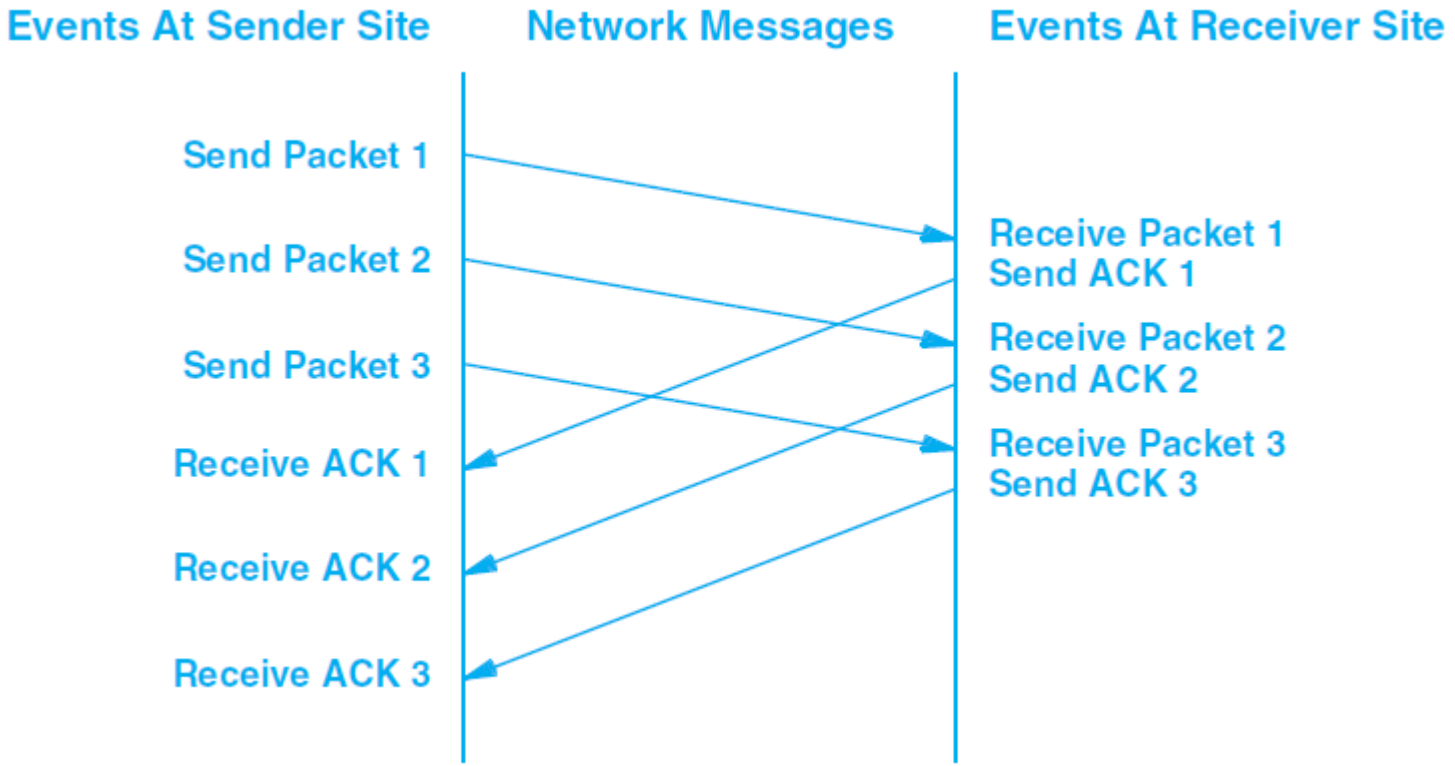
- 利用滑动窗口机制可以保证网络在一定程度的满载，从而提高网络利用率；通过合适的机制，如累积确认，或进一步的，选择确认，则同时可以提高端点的通讯效率：





协议算法
Protocol Algorithms

- 利用滑动窗口机制可以保证网络在一定程度的满载，从而提高网络利用率；通过合适的机制，如累积确认，或进一步的，选择确认，则同时可以提高端点的通讯效率：



协议算法
Protocol Algorithms

- PSTN网络与IP网络相比，最大区别在于PSTN网络的智能在网间，即建立连接之间进行资源预留；而对于IP网络，其智能在端点，尽力而为的数据包传输服务并不知道网间情况；会如果彻底不顾及网络状况，则IP网络可能会出现瘫痪情况。作为网络绝大多数应用的承载协议，TCP协议设计了拥塞控制算法来应对这种情况。

  The dramatic characteristic of PSTN distinguished from IP network is that the intelligence or the awareness of network situation is distributed among the network, which means the resource must be reserved before connection establishment. However, for IP network, the intelligence resides at the end-points, just to provide a best-effort service of delivery. If it is at a all ignorance of the network situation, it is of possibility to bring the whole network down. As the most commonly-used protocol to underpin the majority of applications, TCP incorporates the congestion control algorithm to mitigate extreme case of network traffic overwhelming.

状态机
State Machine

- 状态机是有限状态自动机的简称，其往往是对真实世界事物运行规则，经过逻辑严谨的数学抽象，而形成的一个数学模型。

- 状态机所涉及的概念如下：
  - 状态：即事物或对象在特定上下文中处于的能与其它情况显式区分的情境；通常，一个状态机至少要包含两个状态；
  - 事件：事件就是执行某个操作的触发条件；
  - 动作：事件发生以后要执行动作；编程实现时，一个动作往往会对应于一个函数；
  - 转换：即从一个状态变化为另一个状态。

状态机
State Machine

- State machine is the abbreviation of finite state machine (sometimes also called automata). This mathematical model is usually formed by logically rigorous mathematical abstraction based on the rules of real-world things.
- The concepts involved in the state machine are as follows:
  - State: states indicate an object's situation which can be clearly distinguished from others in a specific context; usually, a state machine must contain at least two states;
  - Event: an event is a triggering condition to perform an operation;
  - Action: an action needs to be executed after an event occurs; An action often corresponds to a function when implementation;
  - Transition: the change of state from one to another.

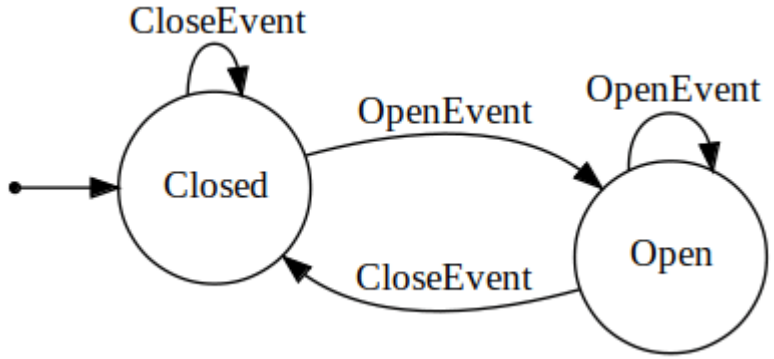
状态机
State Machine

- 状态机的应用
  - 状态机在实际工作开发中应用非常广泛，例如硬件设计，编译器设计，以及编程实现各种具体业务逻辑的时候。
  - 很多网络协议的开发都必须用到状态机；一个健壮的状态机可以让协议程序不论发生何种突发事件，都不会突然进入一个不可预知的程序分支。
- Application of State Machine
  - State machines are widely used in numerous applications, such as hardware design, compiler design, and implementations of various business logic.
  - State machine is intensively used in the implementation of many network protocols; a robust design based on state machine can prevent the protocol program from suddenly entering an unpredictable program branch no matter what unexpected events occur.

状态机
State Machine

- 示例（Example）
  - 我们用一个开关门的情况为例，来说明如何基于状态机进行程序设计。

    We exemplify the design based on state machine based on opening/closing a door.



状态机
State Machine

- 按照面向对象的思想，我们先对状态机进行抽象。显然，状态机应该囊括各个状态的信息，以及当前的状态；同时，我们希望当事件发生时，状态机有一个统一的入口进行处理：

```
template <typename State1, typename State2, typename ... States>
class StateMachine
{
public:
	StateMachine() : currentState(&std::get<0>(states)) {}
	template <typename Event>
	bool handle(const Event& event)
	{
		// …
	};
private:
	std::tuple<State1, State2, States ...> states;
	std::variant<State1*, State2*, States* ...> currentState;
};
```

状态机
State Machine

- 按照面向对象的思想，各个状态自身应该处理当该状态为状态机当前状态时，如果有事件进来，进行何种处理。即状态机仅提供一个总的入口，各个状态自己负责处理对特定事件的响应。因此，上面状态机的类中，对事件处理的函数如下所示：

```
template <typename State1, typename State2, typename ... States>
class StateMachine
{
public:
        // …
        template <typename Event>
        bool handle(const Event& event)
        {
                auto passEventToState = [this, &event](auto statePtr){
                        statePtr->handle(event);
                };
                std::visit(passEventToState, currentState);
                return true;
        }
        // …
};
```

状态机
State Machine

- 按照面向对象的思想，我们需要将各个状态实现为类的实例，例如，对门开着的状态有如下抽象及其实现：

```
class OpenState
{
public:
	bool handle(const OpenEvent& e)
	{
		cout << "Do nothing, since the door is already open!" << endl;
		return true;
	}

	bool handle(const CloseEvent&)
	{
		cout << "Closing the door ..." << endl;
		return true;
	}
};
```

状态机
State Machine

- 但上面实现有个问题，即当处理完事件，要转移到其它状态时，如何通知状态机转移。如果返回新的状态，这是不行的，因为状态机本身已经有了各种状态，返回状态对象必然是一种浪费；另外，此种方式必然涉及到类的前向声明，而前向声明不允许使用构造函数，因此一种想法是返回类型，在状态机机里由类型决定该转移至何种状态：

```
class OpenState
{
public:
	OpenState* handle(const OpenEvent& e)
	{
		cout << "Do nothing, since the door is already open!" << endl;
		return NULL;
	}
	ClosedState handle(const CloseEvent&)
	{
		cout << "Closing the door ..." << endl;
		return NULL;
	}
};
```

状态机
State Machine

- 根据上面思路，进入统一入口之后，我们需要将具体处理委托给具体状态，然后根据返回值的类型（而不是返回值本身），决定状态转移：

```
template <typename State1, typename State2, typename ... States>
class StateMachine
{
public:
	StateMachine() : currentState(&std::get<0>(states)) {}
	template <typename Event>
	bool handle(const Event& event)
	{
		auto passEventToState = [this, &event](auto statePtr) {
			return statePtr->handle(event);
		};
		auto p = std::visit(passEventToState, currentStatePtr);
		currentStatePtr = &std::get<std::remove_pointer<decltype(p)>::type>(states);
		return true;
	};
};
```

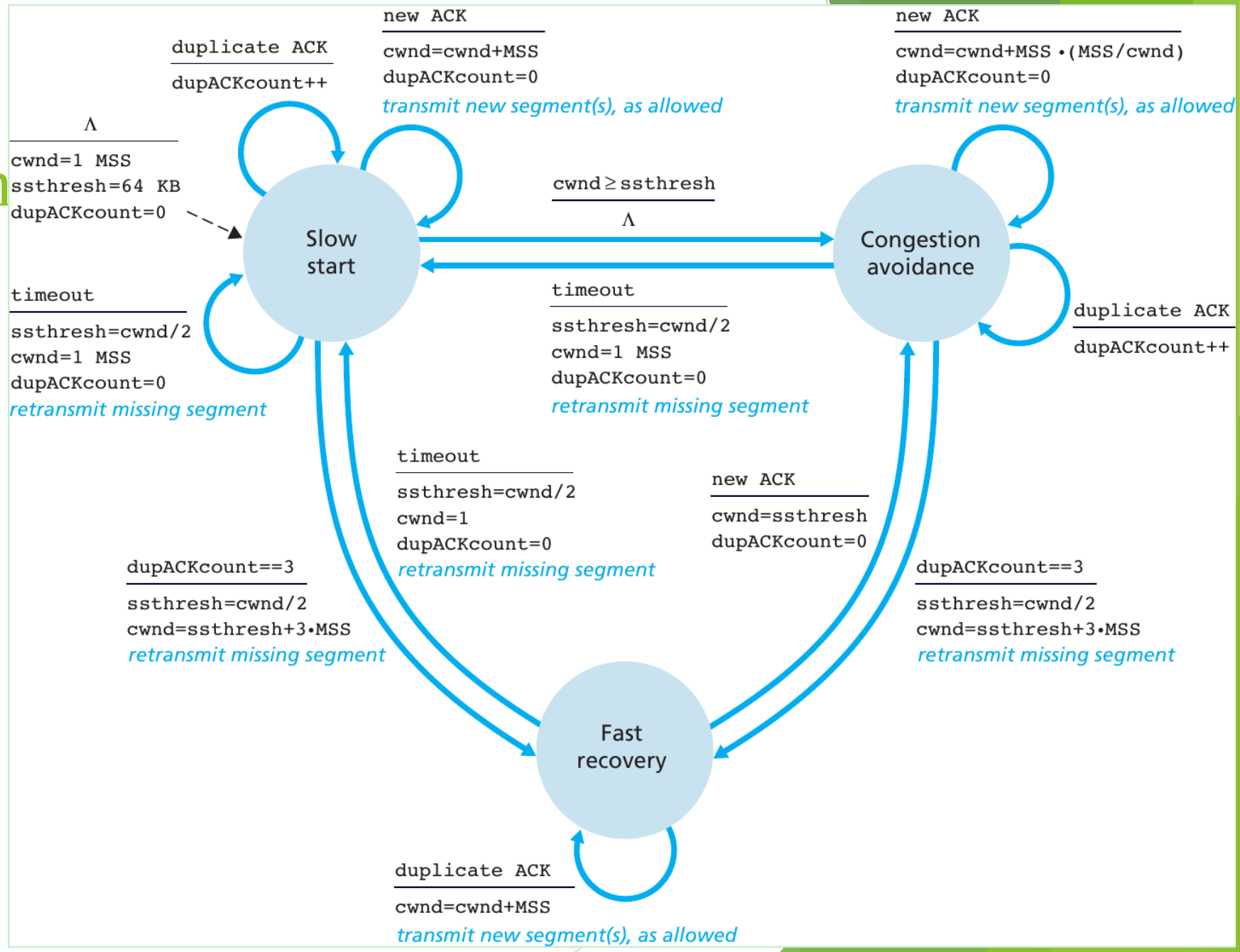
状态机
State Machine

- 根据上面抽象及实现，我们将开关门的状态机实例化及测试如下：

```
using Door = StateMachine<ClosedState, OpenState>;

void sm_door_test()
{
	Door door;

	door.handle(OpenEvent{});
	door.handle(OpenEvent{});
	door.handle(CloseEvent{});

	door.handle(CloseEvent{});
	door.handle(OpenEvent());
}
```

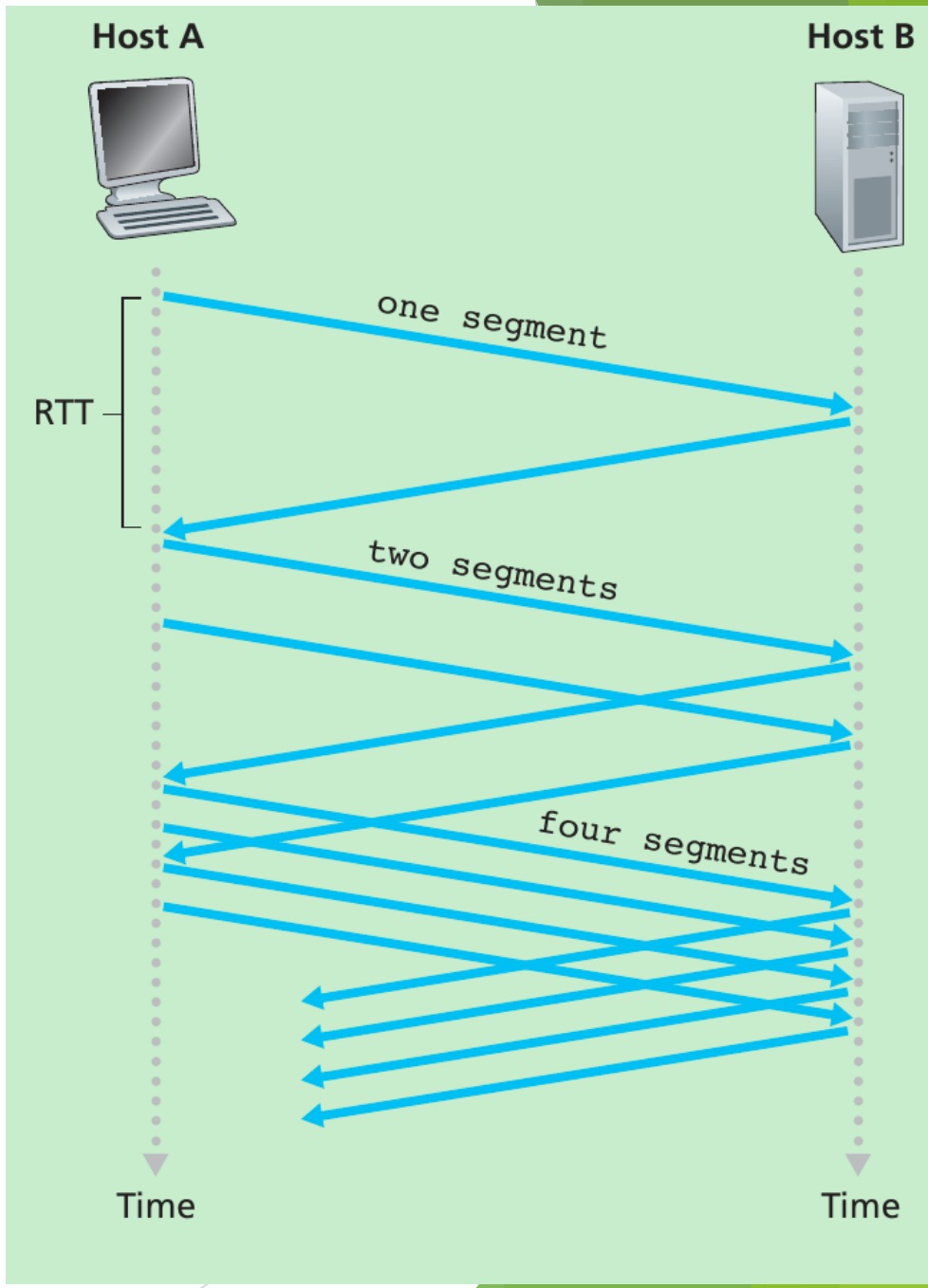
协议算法
Protocol Algorith

- TCP协议的拥塞控制主要是发送端根据与接收端交互的情况，做出的响应；其根据对网络情况的估算，从而处于不同的状态。如下的状态机展示了TCP发送端在处理拥控时，可能所处的状态及经历的状态转变：



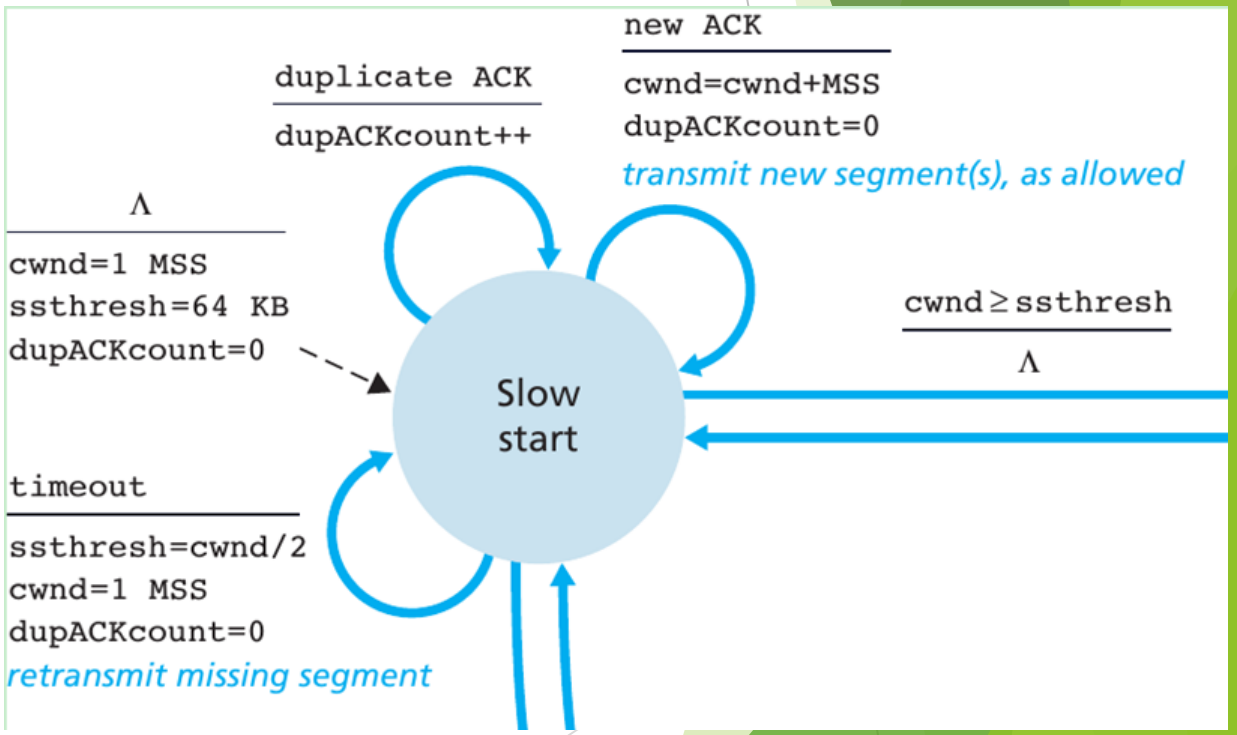
协议算法
Protocol Algorithms

- TCP协议的慢启动阶段如右图所示。对每个TCP连接，会维护一个拥控窗口cwnd，其初始大小最设置成一个最大网络报文段MSS大小。然后，当发送端每收到一个接收端的确认信息时，都会将cwnd增加一个MSS大小。注意，由于是对每个ACK都会都加一个MSS大小，因此cwnd增加实际上是线性的。



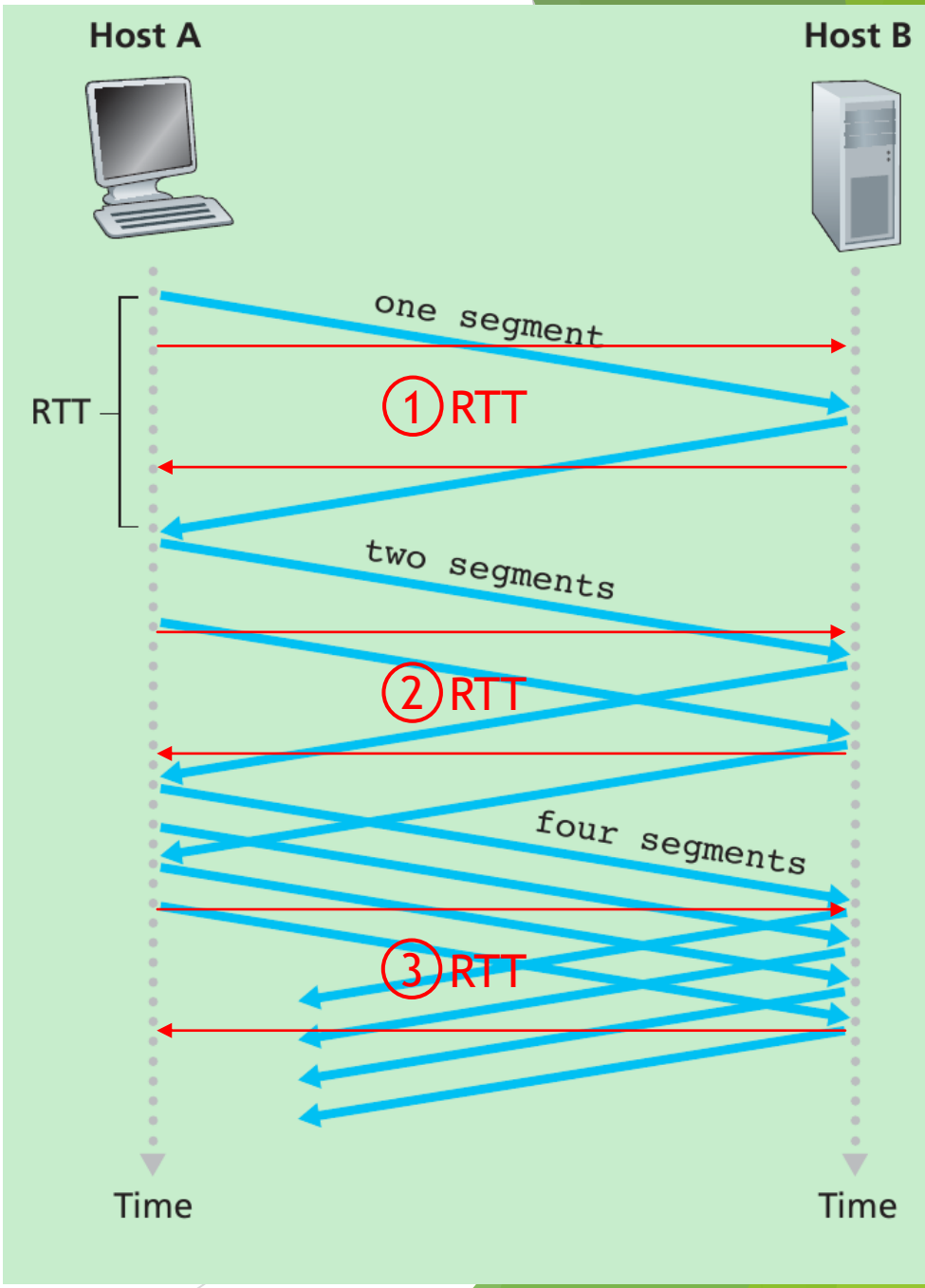
协议算法
Protocol Algorithms

- 在慢启动阶段，如果出现接收对端确认超时，即出现丢包，则会将默认的慢启动阀限值ssthreshold设为当前cwnd窗口的一半，即ssthreshold = cwnd / 2，将cwnd设为1个MSS，重新开始慢启动过程；
- 如果在达到慢启动阀限值之前，收到重复确认的数目超过3，则进入快速重传，随后进入拥塞避免阶段；
- 如果已达到慢启动阀限值并且收到重复确认的数目不超过3，则进入拥塞避免阶段。



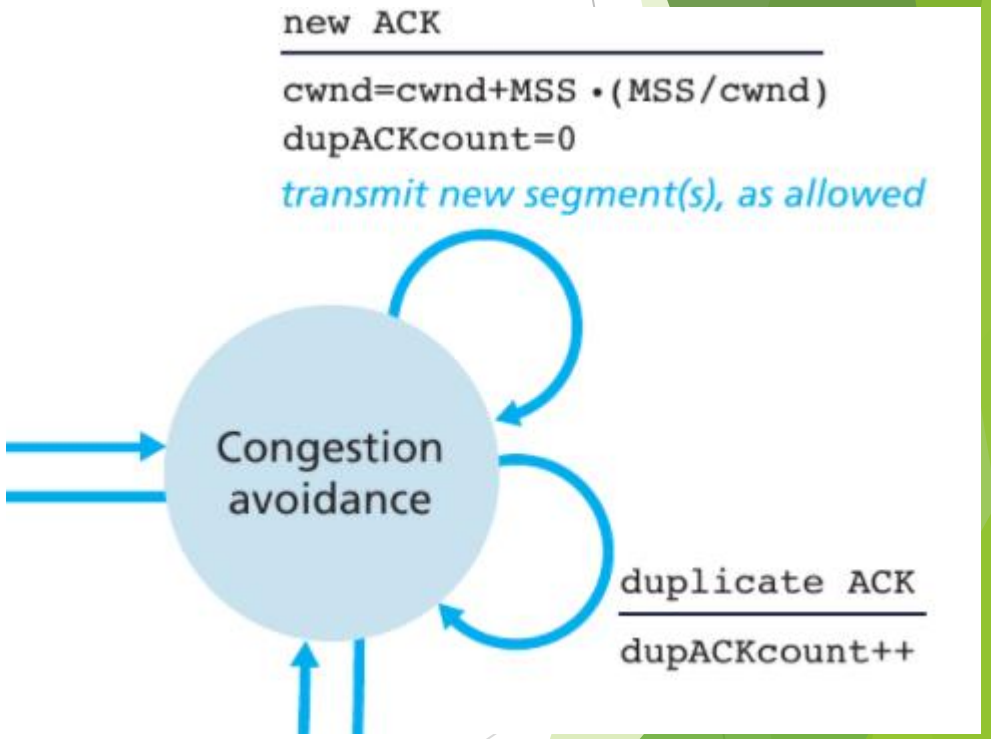
协议算法
Protocol Algorithms

- 在讲拥塞避免状态之前，我们要先明确一个概念，即RTT（Round-Trip Time），其如图所示。在慢启动阶段，我们可以看到，如果对每个收到的ACK都将拥控窗口的大小增加一个MSS，则在一个RTT内，cwnd呈指数级增加。如果我们希望一个RTT内，cwnd的大小仅增加一个MSS，该怎么做呢？注意在①时，一个RTT内收到1个ACK，此时cwnd为1MSS；在②时，一个RTT内收到2个ACK，此时cwnd为2MSS；在③时，一个RTT内收到4个ACK，此时cwnd为4MSS；
- 若每收到一个ACK，将cwnd增加$\frac{\text{MSS}}{\text{cwnd}}$·MSS大小，则我们发现，在①时，收到一个ACK时，cwnd为1MSS，则增加即为1MSS，在②时，收到一个ACK时， cwnd为2MSS，即增加0.5MSS，但前后共收到2个ACK，即1个RTT增加1个MSS；同理可对③进行分析



协议算法
Protocol Algorithms

- 当由慢启动状态进入拥塞避免状态之后，会首先将重复ACK计数清零；同时，如前所述，cwnd增长由每ACK增加1MSS，变成了每RTT增1MSS；
- 如果在拥塞避免状态出现超时，说明网络可能出现了拥塞，这个时候则需重新回到慢启动状态，即将慢启动阀限设置为当前拥塞窗口的一半，将拥塞窗口重新置为1MSS，进行慢启动过程。
- 如果在拥塞避免状态收到的重复ACK不超过3个，仅将重复ACK计数，不进行状态转移；当等于3个时，则将慢启动阀限设置为当前拥塞窗口的一半，拥塞窗口增长模式为每ACK增加3MSS。



协议算法
Protocol Algorithms

- 同理，我们可以根据所学的状态机的知识，对TCP所处的其它状态进行分析；

- 小知识点：
  - 算法是“精确”的，但可能不是“精密”的；
  - TCP的状态机是在内核中实现的，为保证效率，内核函数的调用不会保存浮点数运算相关的栈；此即说明，内核中乘、除都是用整数表示的，特别是除法，只会round到最小的接近整数的值；
  - TCP协议以这种“不精密”的方式，工作得很好。

- 希望通过以上讲解，大家对网络协议涉及的算法，有入门性地了解。

问题
Question

- 请分析TCP协议的四次挥手过程。