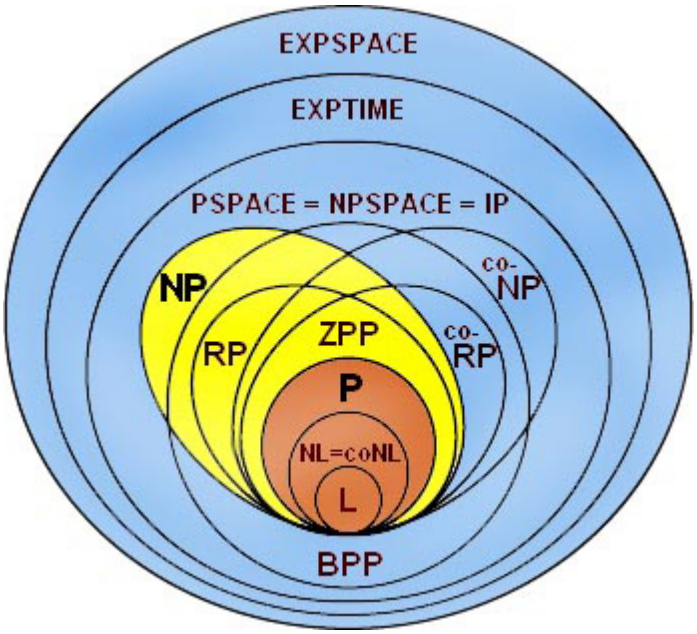
彎曲評論
科技 · 人物 · 潮流

网络系统设计模型概论

作者：KernelChina
kernelchina@gmail.com

编辑：陈怀临
huailin@tektalk.org



序言：

KernelChina撰写的这个网络系统设计模型概论，是在中文环境中难得的一本对通信系统在整体概貌上进行阐述的文献。对初入通信系统的工程师，有志于通信系统研发的学生和研究生人员都是一个很好的读物。编者在其基础上加了一些校注。希望该文对读者有所帮助。这也就是作者和编者，是弯曲评论最大的目的。天行健，君子自强不息；一万年太久，只争朝夕。

网络系统设计模型概论

Pattern是系统设计过程中，重复出现的结构或者原则。不同的设计层次或者领域，有不同的pattern，比如analysis pattern, architecture pattern, design pattern, debug pattern, bug pattern等，甚至还有很多anti-pattern。Pattern是对已有知识的总结和优化，它对现有系统的维护，以及未来系统的设计都有帮助。

在这里，我要总结的是在网络系统设计里面出现的pattern，这只是经验的总结，每一个pattern并没有经过深思熟虑，所以pattern的描述可能是不完整的。而且这些pattern还不能形成一个完整的pattern language，也就是说，它们还可以提炼和扩展。如果能够形成一个pattern language，它将是我写这些文字最大的收获。

Pattern描述需要一定的格式，这里使用的是我自己简化过的描述方式，只保留了最基本的东西。复杂，详细的描述要花费时间和精力，有时间再慢慢扩充。下面就按顺序来描述这些pattern，大概有十多个，将会分几个部分发出来。

1) control plane/data plane
Pattern Name and Classification: control plane/ data plane
Intent: 把转发和协议分离，隔离错误，提高性能
Also Known As: control plane/ forwarding plane
Motivation (Forces):

在网络设备里面，最重要的工作就是转发，大部分流量都是在数据平面处理的。控制平面处理协议。一般来说，数据平面需要处理的是需要转发的流量，而控制平面 处理的是终结在本机的流量。可能还需要加一个管理平面(management plane)，用于配置，管理设备。 很明显，数据平面需要高性能的处理器，而控制平面就稍差一点。但控制平面往往会成为被攻击的重点，所以控制平面更关心协议实现的正确性和安全性。

Known Uses: 网关设备

Related Patterns: service plane

References:

1: Requirements for Separation of IP Control and Forwarding

【编者注：上述控制平面与数据平面的划分，算比较经典的划分，例如数通设备。现在的研发趋势是控制面与数据面是而且只是一种逻辑上的划分，物理上可以共享，可以互通。】

2) first path/ fast path

Pattern Name and Classification: first path/ fast path

Intent: 把慢速路径和快速路径分离，隔离错误，提高性能

Also Known As: slow path/ fast path; exception path/ fast path

Motivation (Forces):

在session-based的网络设备里面，创建session的过程比较复杂，而根据session进行转发的工作则比较简单。所以把创建 session的path分离出来，可以只针对这个部分进行优化。一般来说，在connection rate测试，考验的是first path的能力，而throughput 测试考验的是fast path的能力。

Known Uses: session-based gateway

Related Patterns: service plane

References:

1: How to choose the best router switching path for your network

【编者注：first path通常是指地一个packet来了之后，建session的代码过程。fast path通常是指在有硬件加速的系统中，一个packet来了之后，如果有session的hit，就通过硬件的forwarding迅速的做出policy的决策，而不通过CPU了。。。通过CPU做的读者可以认为是Slow Path。】

3) Queue based design

Pattern Name and Classification: queue based design

Intent: 简化包处理流程

Also Known As: pending and polling

Motivation (Forces):

包交换（存储转发）网络，最基本的操作就是enqueue/dequeue。这个和电路交

换网络有本质的不同。电路交换， 在包通过的路径上，有资源预留，所以不会有等待的情况；而包交换网络是按跳(hop)转发的，在包通过的路径上，没有资源预留，所以当一个包不能被立即转 发时，需要放到队列里面。队列可以分为很多种，比如按包大小分类的队列，按协议分类的队列等等。由于入队和出队是两个独立的操作，所以有多种入队和出队的 策略。网络设备的处理基本上就是：从队列里面取出packet；处理；把packet放入队列。不同的处理可以基于不同的队列，比如在tunnel的处理 过程中，可以把封装后的包放入一个新的队列，后续的处理流程可以从队列里面把包取出来，然后发送出去。引入队列以后，针对不同队列，可以有不同的处理过 程，这些过程相对独立，编程或者纠错都方便一些。

引入队列之后，可能会使得系统的处理时间变长，从而是系统的latency变大。入队，出队操作越多，latency就越大，这一点需要注意。

Known Uses: ip network gateway

Related Patterns: event-driven programming

References:

1: http://en.wikipedia.org/wiki/Queueing_theory

2: Elements of Queuing Systems

3: Event-driven programming

【编者注：有bufer，就有Queue。有Ingress Queue和Engress Queue。有Queue，就有Congestion，就有QoS。Queue的精华就是处理QoS。最简单的QoS就是Packet来了之后，做一次分类（Classify），然后做相应的处理，包括drop。希望读者增加一些QoS的知识。】

4) Per thread queue

Pattern Name and Classification: per thread queue

Intent: 每个thread有自己的队列，减小冲突，增加并行

Also Known As: M model；costco model

Motivation (Forces):

在多核编程里面，任务调度非常重要。如果每个核都是独立poll自己的队列，那么这些核之间就不需要锁。锁只存在于：其他核把包放入接收队列；处理完成之后，把包放入发送队列，由发送核把包发出去。

Receiving core —> Processing core —> Transmitting core

只需要两个锁，而且最多有两个core在竞争这些锁。

这里面的问题在于：Receiving core如何把packet有效分配给processing core，使得每个processing core的workload是均衡的；还有就是如果processing core需要把包放入入队列，它只能放入当前core所在的队列，这样就是的任务处理更加不均衡。

Known Uses: multicore gateway

Related Patterns: Global queue
References:
1: Queueing Theory In Action, plus, frogs
【编者注：在Cisco的QFP系统中，是指前端有一个Packet Dispatcher，负责把一个报文copy到芯片内部的packet buffer中，然后选择和指定一个hardware thread来处理。这种模式的前提是这个判官必须非常公平，和快速的做决定。另外，要能迅速的感系统中的负载情况。】

5) Global queue

Pattern Name and Classification: global queue
Intent: 多个thread共享队列，任务共享，均衡调度
Also Known As: W model; fry's model
Motivation (Forces):

在多核编程里面，如果多个核共享一个队列，每个核都是从这个队列里面把包取出来，然后处理。这样就每个核的load就是均衡的，因为每个核都是平等的，处理相同的任务。

/—> Processing thread —\
Receiving core —> Queue ——> Processing thread —> Queue —> Transmitting thread
\—> Processing thread —/

这里面的问题在于：锁冲突的几率增大；queue管理比较复杂。
Known Uses: multicore gateway
Related Patterns: per thread queue
References:
1: Queueing Theory In Action, plus, frogs
【编者注：全局Queue的编程模型稍微简单。要注意spinlock的使用。全局Queue和每个线程的Queue的区别就类似与超市的排队方式。在大辽国，最典型的就是：Fryes的checkout模型（一个Queue，许多柜台。有一个Dispatcher告诉你去那个柜台checkout【available的时候会亮小绿灯）；Safeway的方式就是许多Queue。你自己看着办。都有优缺点。否则超市的资本家们早就改最优算法了。】

6) Run to completion

Pattern Name and Classification: run to completion
Intent: 在任务处理过程中，禁止调度
Also Known As: No
Motivation (Forces):

在实时系统里面，一个任务开始运行后，它应该被执行完成之后，才能调度其他任务到当前的CPU上运行，也就是说，这个任务不应该不中断，否则的话，任务运行的时间就是无法估计的，而且需要保护的东西将会有很多。网络系统是一个典型

的实时系统，所以在包处理过程中，它是run to completion的。
Known Uses: network system
Related Patterns: pipe line; preemptive
References:
1: Design Patterns for Packet Processing Applications on Multi-core Intel® Architecture Processors
2: More about multicores and multiprocessors
【编者注：RTC对网络系统是一把双刃的刀。关键是看工程师的理解。用的不好，系统很容易starving其他逻辑。在RTC模型下，能看见人性的贪婪。没人愿意轻易yield CPU。】

7) Pipe line

Pattern Name and Classification: pipe line
Intent: 复杂任务需要分解成多个任务，分阶段执行
Also Known As: pipeline and filter
Motivation (Forces):
在网络系统里面，如果一个任务很复杂，需要很多CPU时间，那么这个任务需要分解成多个小任务来执行，否则的话，这个任务占用CPU时间过长，导致其他任务无法执行。而且多个每个核都执行这样的任务，那么，整个系统就无法处理其他任务，这将会导致严重的丢包。所以，每个任务执行的时间不应该有很大的差别，否则，任务调度就有困难。
一个任务分解成多个小任务后，每个小任务之间由queue连接，上一次处理完成之后，放入下一个队列。这样可以任务调度更均衡。每个小任务都是run to completion的，这一点需要注意。
Known Uses: network system
Related Patterns: run to completion; parallel
References:
1: Design Patterns for Packet Processing Applications on Multi-core Intel® Architecture Processors
2: More about multicores and multiprocessors
【编者注：EZChip的编程模型是典型的流水方式。流水模型最大的困难是任务划分粒度的把握。否则，系统的短板就是那个最慢的。通常建议，如果把握不好一个系统，用SuperScalar模型，不用Pipeline模型。想想Intel都不敢做超级流水线了。。。芯片设计，网络设计，都是形不同；而神似。】

8 unreliable message

Pattern Name and Classification: unreliable message
Intent: 快速传递消息，可以容忍少量消息丢失
Also Known As: No

Motivation (Forces):

通过网络，或者通过某种通道传递消息时，需要考虑消息传递是否是可靠的。可靠的消息可以这样定义：

a: 消息是完整，正确的。这个一般用校验和来保证消息在传递过程中没有改变。

b: 接收方一定能够收到消息。接收方收到消息后，先检查校验和，然后给发送方一个应答。发送方在收到应答后，可以确认消息已经送达。如果发送方没有收到应答， 需要定时重传消息，直到收到应答为止。如果消息之间需要保证顺序，那么在前一个消息没有发送完成之前，后续的消息不能发送。

c: 发送方的消息不能丢失。当发送方的应用把消息传递给消息发送模块后，如果它假设这个消息是可靠的，那么在消息模块里面要保证这个消息不能丢失。也就是说， 如果应用模块把消息传递给消息模块并且返回成功，那么这个消息一定能到达接收方，或者消息模块需要返回错误给应用模块。

可以看到，为保证消息的可靠性，需要大量的额外工作，这将会严重影响系统的性能，特别是在高速通道上 更是如此。所以，如果少量的消息丢失是可以容忍的，那么建议使用unreliable message。但是应用程序需要 考虑消息丢失的情况，并为此做相应的处理。

Known Uses: UDP

Related Patterns: reliable message

References:

1: http://en.wikipedia.org/wiki/User_Datagram_Protocol

【编者注: 在高端网络系统设计时，如果底层有高速Fabric，建议通过Special Channel，为控制消息做通道，然后采取非Reliable Messaging。】

9) reliable message

Pattern Name and Classification: reliable message

Intent: 提供可靠的消息通道，简化应用程序的流程

Also Known As: No

Motivation (Forces):

可靠的消息通道可以简化上层应用的处理流程，上层应用不需要为消息丢失或者重复做额外处理。可靠性一般通过两种手段来保证：

a: 确认。收到消息需要确认。

b: 重传。消息和确认都有可能重传，重传需要定时器，也需要队列。

可靠还意味着消息是按顺序送给上层应用的，并且已经去掉了重复的消息。

为了使消息传递得更快，需要使用滑动窗口；为了避免消息通道拥塞，需要采用拥塞控制手段；为保证消息不会使发送和接收双方的缓冲区溢出，需要控制发送和

接 收的消息数量。所以，可靠消息通道的实现非常复杂，面临的难题也很多。如果上层应用普遍要求使用reliable message，那么使用reliable message可以简化应用的设计，否则，可以使用unreliabe message，而由应用来处理丢包和重复的问题。
Known Uses: TCP
Related Patterns: unreliable message
References:
1: http://en.wikipedia.org/wiki/Transmission_Control_Protocol
2: http://en.wikipedia.org/wiki/Reliable_messaging
【编者注：TCP的最大缺点其实就是这方面，特别是在为无线的情况下。TCP最大的假设是丢包=路由阻塞。这在无线网络下是不成立的。】

10) synchronous message

Pattern Name and Classification: synchronous message
Intent: 当后续操作依赖于当前消息的响应时，需要使用同步消息
Also Known As: No
Motivation (Forces):
同步消息的涵义是：在发送请求之后，当前的操作必须等待应答消息返回以后，才能继续后面的流程。这个等待可以忙等待，也可以是睡眠。使用这个模式的前提是：后续操作依赖于当前操作的执行结果，也就是说，这些操作是序列化的。
由于这些操作是序列化的，所以很难并行化，而且不管是忙等待还是睡眠都会耗费一定的资源。所以，除非必要，最好不要用同步消息。
Known Uses:
Related Patterns: asynchronous message
References:
1: http://en.wikipedia.org/wiki/Message_passing
【编者注：在强同步Message方面，对于网络通信，其实是指Causal Ordering。换言之，当两个事件有因果原因的时候，就一定要保证时序的先后。】

11) asynchronous message

Pattern Name and Classification: asynchronous message
Intent: 发送消息后，不用等待应答，消息和应答可以由不同的线程处理
Also Known As: No
Motivation (Forces):
如果一系列操作之间并没有依赖关系，那么消息发送和应答处理可以由不同的线程处理。这样做的好处是：消息发送和应答处理可以并行执行。一般来说，异步消息 并不需要序列化。但是如果有序列化的需求，就需要给消息分配序号，并且使用锁来保证每个消息和应答都是按顺序处理的。在实现时，需要权衡序列化带来的复杂 性，可能用同步消息会有更好的结果。

有个同事把同步消息比做打电话，把异步消息比做电子邮件。打电话需要同步等待，而看邮件就不需要即时响应，也不需要按顺序看。这个类比很形象。
Known Uses:
Related Patterns: synchronous message
References:
1: http://en.wikipedia.org/wiki/Message_passing
2: http://en.wikipedia.org/wiki/Messaging_pattern
【编者注：异步消息的最好理解是想想Unix的Signal。一个进程可以给另外一个进程发一个signal。但目标进程是而且只是在trap kernel之后返回时才会顺便看看。。。胆战心惊的希望没有得到Kill -9】

12) piggyback message

Pattern Name and Classification: piggyback message
Intent: 发送消息时，捎带其他消息
Also Known As: No
Motivation (Forces):
piggyback，意思是捎带，也就是搭顺风车的意思。捎带的好处是可以节省一个或多个消息，对性能有好处。捎带的这个消息可以与当前的消息相关，也可以无关。当然，前提是通信双方有多个消息需要交换，如果只有一个，就没有什么可以捎带的了。
Known Uses: TCP
Related Patterns: No
References:
1: http://en.wikipedia.org/wiki/Piggybacking_(data_transmission)
【编者注：例如，TCP的SYN/ACK message】

13) single message

Pattern Name and Classification: single message
Intent: 每次只发送或接收一个消息
Also Known As: No
Motivation (Forces):
每次只发送或接收一个消息。这样做的好处是简单，不需要花费很多时间在message parsing上面。由于只有一个消息，消息长度和结构的验证都很容易，编程也容易。但一个消息的坏处是：双方交换的消息数量可能会很多，也可能会有其他 开销。以http 1.0为例，每个URL都是独立的，所以需要为每个URL创建一个connection，系统资源消耗较大。
Known Uses: http 1.0
Related Patterns: bundle message
References:

1: http://en.wikipedia.org/wiki/Http

14) bundle message
Pattern Name and Classification: bundle message
Intent: 一次发送多个消息
Also Known As: No
Motivation (Forces):
如果多个消息可以放在一起发送，显然可以减少消息的数目。以http 1.1 tunnel mode为例，一个connection上可以发送多个消息，减少了创建connection的时间，减少了资源消耗。但它会增加message parsing的难度，因为需要它需要确定消息的边界也类型。如果是同一类型消息的bundle还好处理一点，因为长度和类型都是一样的；如果是多种消息 的bundle，就需要使用TLV来定义消息，这更增加了消息处理的难度。
Known Uses: http1.1 tunnel mode
Related Patterns: single message
References:
1: http://en.wikipedia.org/wiki/Http
2: http://en.wikipedia.org/wiki/Type-length-value

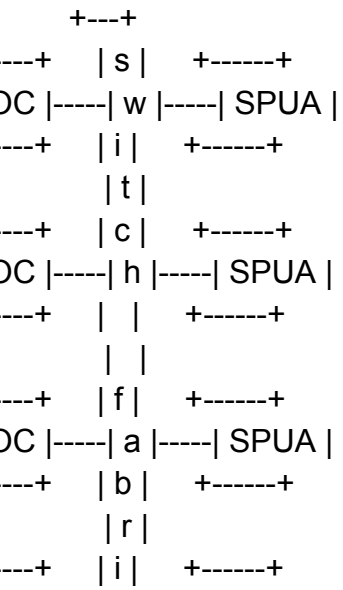
15) p2p
Pattern Name and Classification: p2p
Intent: 一对一的通信管道
Also Known As: unicast
Motivation (Forces):
在网络世界里面，有两种类型的通信管道：p2p和p2mp。在这里，p2p指的是点对点的通信管道，而不是peer-to-peer，peer-to- peer是另外一个话题，在这里不做讨论。p2mp指的是点对多点的通信管道。使用什么样的通信管道和上层应用的需求有关系，比如BGP，使用TCP来交 换协议信息，因为BGP协议一般是在两个AS之间交换路由信息，AS之间的信道本身可能不支持广播，而且不同AS之间交换的信息也可能不同；而OSPF则 使用多播来交换路由信息，因为OSPF一般在内网使用，这个信道通常支持广播，而且一个节点传递给其他节点的信息是一样的。
Known Uses: BGP
Related Patterns: p2mp
References:
1: http://en.wikipedia.org/wiki/Unicast

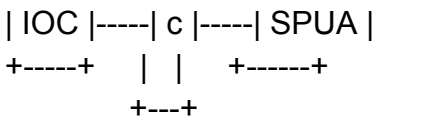
16) p2mp

Pattern Name and Classification: p2mp
Intent: 一对多的通信管道
Also Known As: multicast
Motivation (Forces):
使用p2mp的前提是发送给其他点的信息是一样的，并且最好底层信道支持广播。通常它用于在多点之间维护一个一致的状态机，也就是说，这个状态机在每个节 点上都是一样的，每个节点都会更新自身的状态并把这个变化同步给其他节点。当然，如果底层信道不支持广播，可以使用p2p来模拟p2mp，但是这样做的开 销很大。
Known Uses: OSPF
Related Patterns: p2p
References:
1: http://en.wikipedia.org/wiki/Multicast
【编者注：组播是分布式系统比较难的一个方向。网络背景的人通常误解组播或者简单化组播。组播最大的问题就是Ordering的问题。有兴趣的读者可以阅读一些时序逻辑和分布式系统Group Communication的文章。】

17) centralized packet scheduler

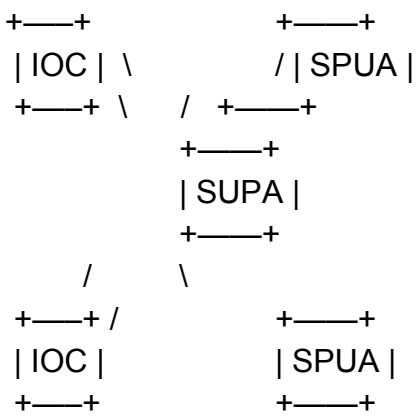
Pattern Name and Classification: centralized packet scheduler
Intent: 把packet或stream动态分配给处理单元
Also Known As: dynamic packet scheduler
Motivation (Forces):
以下四个模式的想法来自于这个帖子：
"Secospace USG9110评测报告"
假设USG9110是这样的结构：





那么如何分发packet/stream？如何管理资源？

一个选择是这样：



其中一个SPUA是特殊的，主要分发packet或者stream，其他SPUA处理业务。这样做的好处是loadbalance可以很好，但坏处也有两个：

1) 这个特殊的SPUA可能会成为瓶颈，导致无法scale out

2) 在SPUA少的情况下，这个特殊的SPUA的性能被浪费了

Known Uses:

Related Patterns: decentralized packet scheduler

References:

1: http://en.wikipedia.org/wiki/Peer-to-peer

【编者注：在这个Dispatcher SPUA可能成为瓶颈的问题里，还隐藏这一个分布式系统的理论问题：在分布式系统中，任何一个节点，在任何一个给定的时候，是不可能知道全局状态的。】

18) decentralized packet scheduler

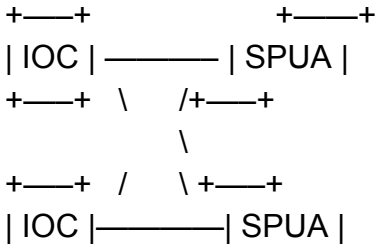
Pattern Name and Classification: decentralized packet scheduler

Intent: 把packet或stream静态分配给处理单元

Also Known As: static packet scheduler

Motivation (Forces):

如何分发packet/stream？管理资源？还有一个办法：



+——+ +——+

IOC上，用哈希算法，把packet/stream直接映射到SPUA上，然后由SPUA处理业务。这样做的好处是可以很好地scale out，但坏处也有两个：

1) load balance不好做。由于是静态映射，很容易受到攻击；在某一个SPUA出现over load时，映射函数无法感知。

2) 热插拔支持比较困难。在SPUA的数量改变时，相应的映射也需要变化，要把这个变化限制在一定范围之内，而不是全局重新映射。

Known Uses:

Related Patterns: centralized packet scheduler

References:

1: http://en.wikipedia.org/wiki/Peer-to-peer

【编者注：据说Hillstone是这样做的。这样的缺点就是IOC需要很智能。估计会比较贵：-）。例如，单独放一个EZChip是比较难的了。估计要加一个CPU了。负载平衡确实难做。如果有快速的，独享的control message channel，IOC对系统的感知就会好很多。否则，一旦发组播packet，会可能把系统折腾跨。。。】

19) centralized resource management

Pattern Name and Classification: centralized resource management

Intent: 集中统一管理资源

Also Known As: dynamic resource management

Motivation (Forces):

把需要动态分配的资源放在某一个SPUA上，统一管理。这样做的好处是资源利用率高，实现简单，但坏处是需要交换的消息太多，影响性能。当然也可以减少消息数量，比如使用bundle message；或者是从上一级分配一块资源，然后在本地管理。本地分配肯定要比远程分配快，这个和cache的性质差不多。

Known Uses:

Related Patterns: decentralized resource management

References:

1: http://en.wikipedia.org/wiki/Shared_resource

2: Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management

20) decentralized resource management

Pattern Name and Classification: centralized resource management

Intent: 把资源静态分配给各节点，由各节点自主管理各自的资源

Also Known As: static resource management

Motivation (Forces):

先把资源静态分配给每个SPUA，然后由每个SPUA自己管理自己的资源。如果配

合decentralized packet scheduler使用，可以很好地scale out，但问题也不少。一是资源浪费（这个和load balance不平衡有关），再就是资源重新分配（比如在热插拔的情况下）。scale out的设计是最好的，但是难度也相当大。
Known Uses:
Related Patterns: centralized resource management
References:
1: http://en.wikipedia.org/wiki/Shared_resource
2: Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management
【编者注：集中还是分布，还是分布的集中，其实都是折中。总体把握是在性价比合适的情况下，系统越简单越好。在以后Sandy Bridge的x86冲击下，QPI+PCI-E 3，确实可能使得分布式的集中资源分配管理变得简单化。其实RMI 832的ccNUMA解决方案也在朝着这个方向演变。但ccNUMA能否有效的用在通信系统中，是一实战考验，而非理论探索。】

21) SMP

Pattern Name and Classification: Symmetric multiprocessing
Intent: 合理分配计算资源
Also Known As: cluster(parallel) model
Motivation (Forces):
SMP/AMP主要关注在多核环境中核的分配。multiple processor是和single processor相对应。
而symmetric和asymmetric相对应。不管symmetric还是asymmetric，它们都面临相同的问题:

在多核下编程。从字面意义来讲，symmetric就是说所有的core做的事情是一样的，这并不是说在任何时刻，每个核的状态都是一样的，而是说，每个core完成的功能是相同的，不存在差别。而asymmetric与之相反，每个core完成的功能是不同的。当然在实际使用时，应该是两种模式混合使用，也就是说，可能从整体来看，系统是AMP，而从某个局部来看，系统是SMP。

使用SMP还是AMP，与系统所面临的任务相关。比如在一个包处理过程中，可能包含以下步骤
a) dequeue，把包从网卡的buffer里面取出来，并放到系统buffer里面。
b) parse，parse这个包，取出感兴趣的信息。
c) classify，包分类。
d) process，处理包。

e) enqueue, 把包放到发送队列。
可以在一个core上把这些事都做了，也可以一个core或者一些core做某一个步骤。
分配的原则是
a) 每一个stage的cpu load是平衡的，也就是说，复杂的事情，可以多用几个核，而简单的事情可以少用几个核。如果是SMP，也就是说，每个核做的事情一样，那么会出现这种情况：所有的核都在忙着处理某一种事情，而不能处理其他事情，这样会导致丢包。所以要避免某一个任务耗费时间过长，而导致整个系统处于忙等待状态。这里的任务是协作式的，所以要在设计时避免长任务。
b) core与core之间有共享的数据结构，因此要使用锁。但是某些任务使用SMP会增加难度，比如收包，发包和保序等。一个core收包和多个core收包，哪个更好需要测试才能确定。并不是核越多就越好，需要在复杂度和功能，速度几个方面来权衡。
Known Uses:
Related Patterns: run-to-completion, pipe line, AMP
References:
1: http://en.wikipedia.org/wiki/Symmetric_multiprocessing
2: http://en.wikipedia.org/wiki/Asymmetric_multiprocessing
3: Hard Real-Time Performances in Multiprocessor-Embedded Systems Using ASMP-Linux

22) AMP
Pattern Name and Classification: Asymmetric multiprocessing
Intent: 合理分配计算资源
Also Known As: pipe model
Motivation (Forces):
具体的讨论可以参见SMP。真实环境里面用到的一般都是AMP和SMP的混合体，所以这两个模式不是竞争，而是合作的关系。AMP一般用于划分control plane和data plane，但是在data plane，也会同时用到AMP和SMP，而在control plane，会用SMP (单核就是退化的SMP)。总之，具体问题具体对待。
Known Uses:
Related Patterns: run-to-completion, pipe line, AMP
References:
1: http://www.windriver.com/products/platforms/network-acceleration/
【编者注：SMP vs AMP。一切都是折中。最重要的是底层是一个单一的OS环境。单独把某些核上运行一个bare mental的RTOS的年代基本上过去。目前看不出来有什么问题不能通过类似Zeor Overhead Linux/Freebsd不能解决的问题。如果在性

能损失10%的情况或（和）上限的情况下，系统设计应该不要introduce proprietary OS。弊大于利 if otherwise。另外，感觉业务部署方面，还应该是AMP的模型，但底层OS是一个SMP的结构。】

23) stateless packet processing

Pattern Name and Classification: stateless packet processing

Intent: 简单，快速处理packet

Also Known As: stateless packet filter

Motivation (Forces):

无状态的包处理应该是一个很自然的选择，因为ip协议就是一个无状态的协议。无状态意味着每个包都是独立转发的，相互之间没有关系。无状态和有状态的主要 区别在于包是否匹配session或者是flow（session和flow这两个词在大多数应用场景里面所指的东西是一样的，所以后面直接用 session来代替了，读者明白这个意思就行）。session是路由，策略，MAC地址的cache，这个和“一次路由，多次交换”的机制是一致的。 在无状态的处理过程中，每个包都需要查找路由，匹配策略，查找出口的MAC地址等；而在有状态的处理里面，如果存在session，先查找session；如果查找失败，再查找路由，匹配策略，然后创建session。

无状态好处是在路由，策略，MAC地址等改变时，可以快速感知，而且可以做到per-packet的ECMP。但是无状态对某些应用是不可能实现的，比如 NAT和IPSEC。NAT通常是session based，因为它要求同一个session的packet，转换后也是同一个session；IPSEC的每个包都是编号的，而且SA本身也要求有状 态。从匹配效率来说，policy/acl/route 匹配并不一定比session匹配效率低，看具体应用。如果是单纯的包转发，当然是无状态处理好；如果要加上更多的service，比如netflow,nat, ipsec等，就需要有状态的处理。

Known Uses: router, switch, packet filter firewall

Related Patterns: stateful packet processing

References:

1: https://www.mikrotik.com/testdocs/ros/2.9/ip/flow_content.php

2: http://www.multicorepacketprocessing.com/

24) stateful packet processing

Pattern Name and Classification: stateful packet processing

Intent: 基于session或者flow的包处理

Also Known As: stateful packet inspection

Motivation (Forces):

stateful packet inpsection (SPI)是防火墙技术的一次飞跃。早期stateless packet filter防火墙，在处理动态协议时碰到了问题。而Checkpoint发明的SPI技术通过动

态创建session来解决这个问题。所以 stateful packet processing首要任务是创建session或者flow 。什么是session？session是cache；是什么的cache？路由，策略，MAC地址等。能够匹配session，就说明相应的流被允许通 过，并且在这之上，有一系列处理。session一般是用五元组（协议，源地址，目的地址，源端口，目的端口）匹配的。在处理动态协议的时候，可能还需要 用到expect（请参见linux netfilter），expect是一个不完整的session，在匹配expect之后，可能还需要匹配策略以确定expect是否可以转化成一个 session。一句话，session是stateful packet processing的基础。

stateful的好处是什么？一是有些应用必须是stateful的，比如前面提到的NAT，IPSEC，以及ALG等。stateful的坏处是什么？在系统状态变化时，比如路由，策略，MAC地址变化，维护session的状态比较困难。

Known Uses: stateful packet inspection firewall
Related Patterns: stateless packet processing
References:
1: http://en.wikipedia.org/wiki/Stateful_firewall
2: Netfilter IPsec processing
3: http://netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html
4: http://www.soapatterns.org/stateful_services.php

【编者注：session也可以理解为以状态机。在系统中会有一个期限被ageout。session方面要注意的是layer-7的session处理与网络层的是否无缝合一。Palo Alto Networks的系统号称有一个AppID，能做到7层的session无缝对接。估计是市场口号大于技术创新】

25) per-packet service

Pattern Name and Classification: per-packet service
Intent: 单包处理
Also Known As: packet-based service
Motivation (Forces):

不管是无状态还是有状态的包处理，有些应用是针对单包的，比如NAT，IPSEC。当然，在处理之前，这个包在IP层已经完成了分片组装。对基于UDP的 协议来说，per-packet的处理是很自然的，但是对基于TCP的协议来说，由于协议的边界并不是IP包，所以per-packet的处理就不适用 了。当然，对基于TCP的协议应用per-packet处理在很多情况下也能工作，只是不能适用于所有情况，特别是包在TCP层被分片的情况。

Known Uses: NAT, IPSEC etc
Related Patterns: stream service
References:

1: http://en.wikipedia.org/wiki/Deep_packet_inspection
2: Per Packet Load Balancing

26) stream service
Pattern Name and Classification: stream service
Intent: 协议的边界是基于流的，需要针对流进行处理
Also Known As: stream-base service, proxy-based service
Motivation (Forces):
基于TCP的协议，上层应用看到的是一个字节流，而不是单个的packet。协议的边界是基于流的，而不是基于单个包。所以，必须先完成流组装才能进行更 高层的处理。协议是分层的，每一层做好自己的事就可以了。stream可以算一个层次，在这之上，有很多应用协议，比如http，还有很多基于http协 议的应用协议，所以可能还需要一个http协议层。这样，每一层只关注自己的工作，向上层提供相应的服务。
Known Uses: DPI, URL filter etc
Related Patterns: per-packet service, tcp-proxy
References:
1: http://en.wikipedia.org/wiki/Flow_(computer_networking)
2: http://en.wikipedia.org/wiki/Data_stream_mining

27) service plane
Pattern Name and Classification: service plane
Intent: 多种service组成一个有机的整体
Also Known As: application layer
Motivation (Forces):

(http://www.nsfocus.com/1_solution/1_2_10.html, 这张图来自绿盟的网站，如有版权问题，请联系本文作者)

如上图，在有这么多service运行的情况下，如何把这些service组成一个有机的整体？ 是callback based, event-driven, pool-based，或者其他？这些都是细枝末节，最重要的是把service的层次要定义出来，这样才能和传统的data plane区分开来。service这么多，调用顺序是一个问题；如何在规定的时间内完成处理是另一个问题（realtime系统的要 求）；service之间如何通讯，service如何管理，service如何配置等等。netfilter做的完全动态，可扩展的框架就很不错，不过 性能稍差一点，可以用做参考。

Known Uses: multi service gateway

Related Patterns: per-packet service; stream service

References:

1: http://www.nsfocus.com/1_solution/1_2_10.html
2: http://en.wikipedia.org/wiki/Software_framework
3: http://en.wikipedia.org/wiki/Osgi
4: http://en.wikipedia.org/wiki/Netfilter

【编者注：service plane是一个比control和data plane对系统更好的理解。编者曾经提出过在系统设计方面如下概念和模型：在control中有control domain和data domain；在control中有fast path和slow path；在data中也有control domain和data

domain；在data中也有fast path和slow path。一个好的系统设计是一个有效的，折中的部署。】

28) tcp-proxy

Pattern Name and Classification: tcp-proxy

Intent: 实现stream service

Also Known As: application proxy

Motivation (Forces):

把tcp-proxy单列出来，看似有凑数之嫌，其实不然。我们常见的application proxy，比如squid，apache等，是应用层的代理，和这里所说的data plane的tcp-proxy完全不同。以squid为例，它完成代理，需要建两个socket，一个包进出内核，需要两次拷贝，这样的开销是不能容忍 的。如果忽略内核空间与用户空间之间的内存拷贝，这个tcp-proxy应该能提供哪些功能？首先要支持多种应用协议，其次需要支持多种 service（多个service，一个tcp-proxy），而且需要支持多个tcp-proxy同时运行（有可能是per-session的 tcp-proxy）。tcp-proxy是TCP协议的一个完整实现，能够工作的实现已经不容易了，高性能的实现是很困难的事情，是业界的难题之一。

Known Uses: multi service gateway

Related Patterns: stream service; service plane

References:

1: http://en.wikipedia.org/wiki/Proxy_server

2: http://en.wikipedia.org/wiki/Squid_proxy

29) Trace

Pattern Name and Classification: Trace

Intent: 提供有效的了解系统状态的工具

Also Known As: Debug; Log

Motivation (Forces):

在系统出问题的时候，如何有效的跟踪定位问题？答案就是有效的Trace系统。这里所说的Trace和通常在Debug image里加的Debug信息是不一样的，在Debug image里面的Debug信息，在Releaseimage通常会通过开关去掉；而这里所说的Trace是指在Release image里面所包含的Trace。包括对关键路径和出错路径的记录。这里的Trace正常情况下应该是关闭的，但是可以通过开关来打开。有了这样的工 具，定位错误就会方便很多。

系统出错并不可怕，可怕的是不能快速定位并解决问题。所以系统一般都会包含一个Trace系统，也会包含一个在系统崩溃情况下，保存寄存器，调用栈，内存等内容的工具。这些是集成在Release image里面的，出错的时候可以帮助定位问

题。
Trace并不是越多就越好，最好是能够理清楚系统有多少个调用路径。一般来说，data plane的调用路径都不会太复杂，应该还是可以理清楚。Trace太多了，反而找不到有用的信息；太少了，又得不到有用的信息。由于Trace是在Release image里面，太多了也会影响系统性能。所以最好是对每一个Trace都经过Review才加进来，尽量避免多加或者少加。
Known Uses: network system
Related Patterns: counter
References:
1: http://en.wikipedia.org/wiki/DTrace
2: http://en.wikipedia.org/wiki/SystemTap

30) Counter
Pattern Name and Classification: Counter
Intent: 记录系统已发生的事件的数量
Also Known As: No
Motivation (Forces):
Counter记录了系统已发生事件的数量，比如收发包的个数，处理出错的个数和原因，这些对了解系统状态很有帮助。Counter简单来说有两类：一是 系统正常处理的事件数，这类Counter可以帮助了解系统是否正常工作；二是系统异常处理的事件数，这类Counter可以帮助了解系统发生了什么问 题。Counter并不是越多越好，特别是在Multi-core的环境里面，atomic counter对系统性能影响很大。Counter的性能是设计系统是需要特别考虑的问题。并不是所有的Counter都需要atomic update，如果可能的话，可以用per-cpu的counter，或者使用普通的update。
Known Uses: network system
Related Patterns: Trace
References:
1: www.cc.gatech.edu/~jx/8803DS08/**counter**.pdf
2: http://www.cc.gatech.edu/classes/AY2010/cs7260_spring/plentycounter-ancs2009-slides.pdf

31) Performance monitor
Pattern Name and Classification: Performance monitor
Intent: 跟踪系统的性能变化
Also Known As: Performance tracker
Motivation (Forces):

在产品的生命周期里面，维持或者提高系统性能是很重要的一个工作。黑盒测试可以了解系统的性能变化趋势，但是不能告诉开发者系统性能为什么发生变化。这个 时候，就需要集成的performance monitor来跟踪系统性能变化。performance monitor应该只针对主要路径，主要函数。因为performance monitor本身会耗费一定的时间。如何把performance monitor集成到release image并使得performance monitor对系统性能的影响最小，需要仔细设计。performance monitor有时也可以帮助开发者了解系统的热点，这样有助于性能优化。
Known Uses: network system
Related Patterns: No
References:
1: *OProfile* – A System Profiler for Linux
2: Smashing performance with *OProfile*
【编 者注：trace和counter从更大的层面理解是系统设计中应该有一个Debug / Profiling Plane。这个逻辑上的平面可以是与control domain合并在一起的。但在系统设计理解中，是一个单独的部分。End2End的Debug和Profiling是把握一个系统非常关键的部分。需要格外重视。这种debug和profiling机制可以包括但不局限于ASIC，NP，CPU，Fabric，报文整个生命周期的各个环节。系统设计时要能够定义出各个子系统，子环节的loopback。从而为动态定位和调试一个系统打下良好的基础。】
32) High availibility
Pattern Name and Classification: High availibility
Intent: 设计具有高可靠性的系统
Also Known As: reliability
Motivation (Forces):
High availibility是系统设计必须面对的问题。所谓的high availibility，就是那些99.9%, 99.99%, 99.999%,99.9999%。没有100%可靠的系统，除非系统不用，或者只用一次，用过之后就不再用了，否则的话100%是没法测试的。在电信领 域，一般都用一年宕机多长时间来衡量系统的可靠性。如何保证系统的可靠性哪？最直接的就是系统冗余，不管是硬件还是软件，提供冗余。冗余包括某些部件的冗 余，也包括整个系统的冗余；有双机的冗余，也有多机的冗余。
引入High availibility的概念后，很多软件设计就需要做相应的变化，比如说配置同步，状态同步，NSR，ISSU等。虽然系统本身是冗余的，但呈现给用 户的还是单一系统，这就是high availibility所面临的难题，用户不需要了解，也不能感知切换，为了这个目标，需要做很多工作。
Known Uses: network system
Related Patterns: fault tolerance

References:
1: http://en.wikipedia.org/wiki/High_availability
2: http://en.wikipedia.org/wiki/High-availability_cluster
【编 者注：HA是通信系统比较复杂的一个部分。但比较限于高端系统。在中低端系统设计中，不需要做首先考量。在路由系统中，要立即NSF，NSR和ISSU等各种概念和机制的区别。要立即Active / Passive，Active / Active系统设计的层次关系。在路由系统中，HA最重要的问题是如何使得路由convergence的性能最好。这也是路由系统设计中control domain最难的问题之一。Aristanetworks的软件系统在高可靠性方面非常有特色，是大亮点。有兴趣的读者请阅读相关资料。】

33) Fault tolerance

Pattern Name and Classification: fault tolerance
Intent: 考虑各种可能性，使系统避免崩溃，能够正常工作
Also Known As: stablility; robust
Motivation (Forces):

Fault tolerance与high availibility不同。简单地说，fault tolerance是high availibility的基础，但是faulttolerance并不能保证high availibility，是必要条件，而不是充要条件。Fault tolerance有很多方面，最简单的，系统要能处理收到的任何包，这里的处理也包括丢弃。畸形包是最常见的攻击之一，如果不能正确处理，就会导致系统 崩溃，或者系统的功能错误。

Be conservative in what you do,
be liberal in what you accept from others.
(http://graphcomp.com/info/rfc/rfc1812.html)

避免崩溃是一方面，系统的功能正确才是最终目的，任何异常处理都不能损害系统的功能，否则就是在功能定义阶段没有把功能定义清楚。
Known Uses: network system
Related Patterns: high availibility
References:
1: http://en.wikipedia.org/wiki/Fault-tolerant_design
2: http://en.wikipedia.org/wiki/Fault-tolerant_system
【编 者注：容错技术在通信系统中不太着意强调。但在服务器系统中，是至关重要的。例如，常说的RAS（Reliability，Availability和Servcability）。 通信系统中，往往是一旦出现状态机出错，系统基本上面临reboot的命运。而服务器领域，是需要带伤做战；许多高级技术，例如进程迁移，虚拟机分区等等。这也是为什么说服务器领域其实是凝聚高端计算技术的前沿。属于mission critical的范畴。 】

34) Watchdog

Pattern Name and Classification: watchdog

Intent: 定时监控，避免某一任务占用CPU时间过长
Also Known As: timer; keepalive
Motivation (Forces):

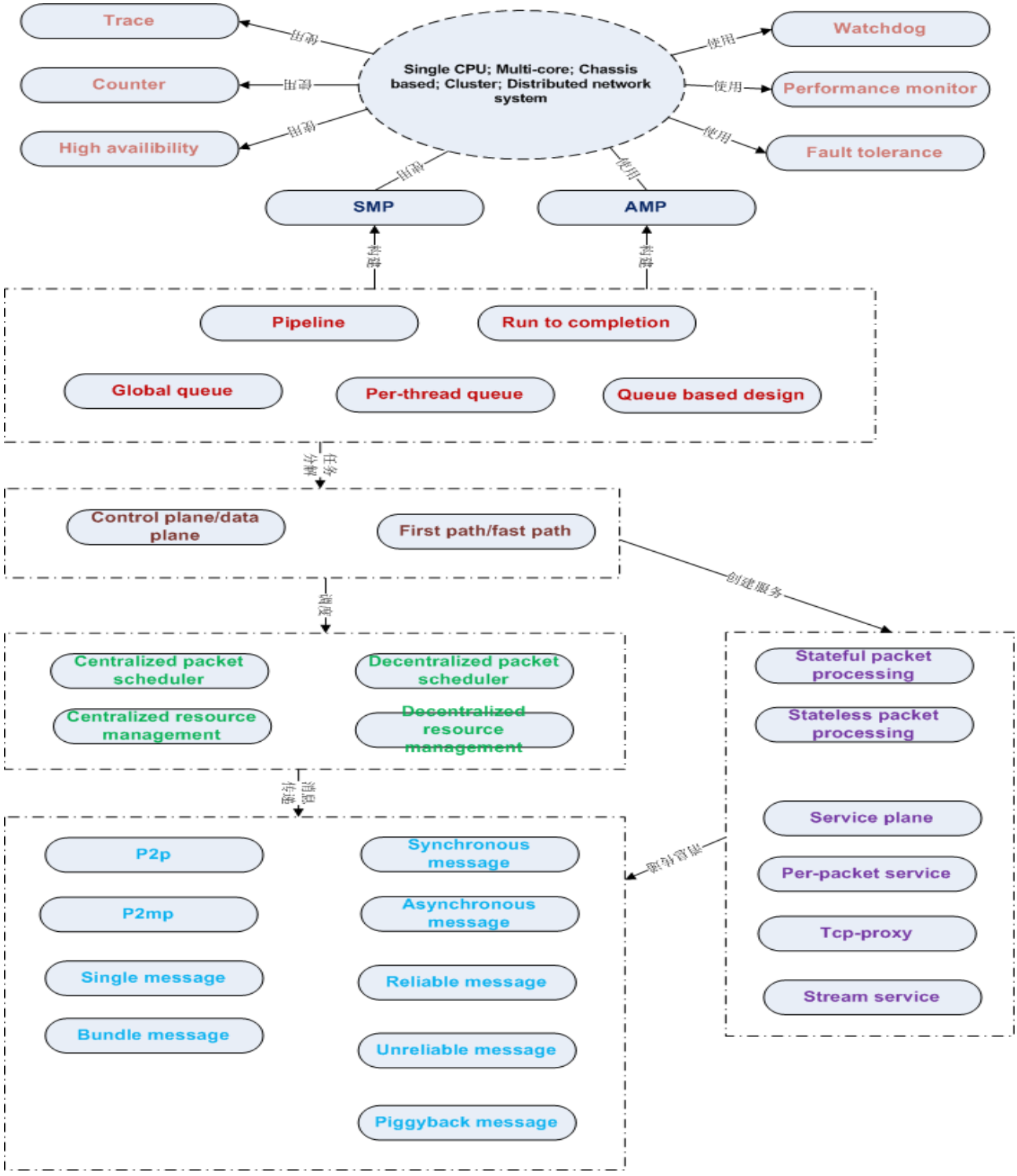
一般来说，网络系统是实时系统，任何一个任务，都应在规定的时间内结束，否则就是系统错误。所以需要watchdog来监控每个任务。watchdog还 可以帮助开发者发现系统中的死锁，过长的循环，任务分配不合理等问题。如果某一任务执行时间过长，它就会阻塞其他任务，如果所有的CPU都被这类任务占用 了，系统就无法相应事件，也有可能无法将这些任务调度出去。

有些需要定时执行的任务，比如HA里面的heartbeat，IPsec里面的DPD等，与watchdog类似，用途也差不多。

有些公司拿watchdog来复位整个系统，当然这个watchdog的粒度就比较大了，和前面说的watchdog不一样。在某些情况下（比如无法定位的错误），必须重启机器才能使系统正常工作，这个办法还是可以用一用的，当然这是最后的选择，不要轻易使用。

Known Uses: network system
Related Patterns:
References:
1: http://en.wikipedia.org/wiki/Watchdog_timer
2: http://en.wikipedia.org/wiki/Keepalive

35) 总结



Single CPU, Multi-core, Chassis based, Cluster, Distributed 的网络系统设计，是软件和硬件co-design的过程，硬件方面，诸如ASIC, FPGA, Co-processor, Network processor, Switch fabric, Memory controller等问题，不在本文的讨论范

围之内，本文作者也不太了解这方面的知识。但从软件方面来说，首先要面对的是如何分配计算资源：SMP或者 AMP，这就涉及到操作系统选择（开源操作系统很多，但都不是为网络设备优化过的，网络设备也是一个嵌入式设备，但是协议栈是通用协议栈，用来做网络设备 可能稍差点），编程方式选择（在内核还是在用户空间），API（标准API，还是自己私有的API），程序库等等。然后是任务分配，任务分配的难点在于如 何让系统的资源使用达到平衡，避免有些CPU太忙，而有些CPU太闲，在这里，Queue做为连接不同任务的节点就显得很重要。在接下来就是软件层次的划 分，无状态包处理还是有状态包处理的选择，最后就是多CPU（chassis based，cluster, Distributed）的通讯，有很多库可以参考，比如MPI或者OpenMP。

当然，做为一个完整的系统，肯定还有很多东西需要考虑，比如内存管理，timer, ager，链表，哈希表，树等等具体的问题，这些都是细节，对系统的整体设计应该没有什么太大影响。硬件设计需要考虑系列化，可扩展；软件设计需要考虑通 用，可移植，可扩展。从单CPU到多CPU，从单板到多板，再到多机。硬件可以scale out，软件同样可以scale out，这样的系统才是成功的系统。不管什么样的系统，从更高层次来看都是一样的，所以google的high performance web和cisco的high performance network，上层结构可能是一样的，这个要慢慢体会。

References（只列出了看过的，翻过的，见过的，其他的没有了，读者可以自己补充）

1: Network Systems Design Using Network Processors: Intel 2XXX Version
2: Network Systems Design with Network Processors, Agere Version
3: Network Flows: Theory, Algorithms, and Applications
4: Patterns in Network Architecture: A Return to Fundamentals
5: Pattern-Oriented Software Architecture Volume 1: A System of Patterns
6: Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects
7: Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management
8: Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing
9: Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages
10: Design Patterns: Elements of Reusable Object-Oriented Software
11: Network Algorithmics,: An Interdisciplinary Approach to Designing Fast Networked Devices (The Morgan Kaufmann Series in Networking)
12: Network Routing: Algorithms, Protocols, and Architectures (The Morgan Kaufmann Series in Networking)

13: Designing Embedded Communications Software

【编 者注：系统设计是考验一个系统架构师的试验场。真正意义上的架构师必须是从实践中成长起来的，但有能具备良好抽象能力的人才。脱离实践的架构是胶片层面的架构；脱离抽象的架构是缺乏创造力的实现。缺一不可。这其中之奥秘就是折中。折中的精华就是对风险的把握和对支撑研发团队能力的判断。激进和保守都没有对错。任何一个技术观点都有pros和cons。唯一的对错就是能否审时度势。 】