摘 要

3G 系统将提供真正的话音、数据和图像的综合移动服务，3G 系统最方便在完全基于 IP 协议的网络平台上实现，其中无线信道的带宽是整个无线通信系统中最宝贵的资源。然而，由于层次性传输协议框架中各层协议的封装，浪费了大量的带宽。为了能在无线链路上获得较好的性能，IETF 工作组提出了 ROHC（RObust Header Compression）信头压缩协议。

本论文首先简要介绍了 ROHC 协议的原理，着重对 ROHC 的三种压缩状态、三种工作模式及其 W-LSB（Windows-Least Significant Bits）算法等进行了阐述。然后给出了 TCP 信头字段的分类方式，按照 ROHC 简档规范，编写了 TCP/IP 压缩的简档。提出了滑动窗口宽度的动态调整方法以适应无线链路的变化特性，并且对提出的方法进行了性能仿真和分析。最后，为了方便 ROHC 的 TCP/IP 压缩简档的编写和测试，设计了 ROHC 的测试信源，在不需要建立连接和确认分组反馈的情况下，测试分组可以持续发送。

关键词：ROHC, W-LSB, SWW, 简档

Abstract

The third generation (3G) mobile system is a true mobile service convergence of voice, data and image. Actually, the only way to make the 3G mobile system convenient seems to be a network platform totally based on IP protocol which includes wireless links, whose bandwidth is the most precious resource of the whole wireless system. However, the encapsulation process of the hierarchical transmission protocols wastes a substantial part of the wireless bandwidth. RObust Header Compression (ROHC) scheme has been proposed by the IETF working group, which aims at providing a compression scheme that has high compression efficiency and high robustness when used over wireless links.

The basic principle of ROHC protocol is introduced, and its three types of the states, three types of operation modes and W-LSB algorithm are expounded especially. The classification of TCP header fields is proposed, in compliance with ROHC classification criteria. TCP/IP header compression profile is given, the rules perfectly follow the guidelines of RFC3095, making the integration of the proposal suitable for the general ROHC framework. The scheme of adjusting slide window width is suit for wireless links, the performance of adjusting slide window width are simulated and analyzed based on TCP/IP application. Finally, ROHC testing source is desgined for checking TCP/IP compression profile, the method is flexible that it can carry on generating test packets without connection or receiving ACK packets.

KeyWords: ROHC, W-LSB, SWW, Profile

南京邮电大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及所取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京邮电大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已

在论文中做了明确的说明并表示了谢意。

研究生签名：李锦中 日期：2009.4.10

南京邮电学院学位论文使用授权声明

南京邮电大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其它复制手段保存论文。本人电子文档的内容和纸制论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权南京邮电大学研究生部办理。

研究生签名：李锦中 导师签名：[illegible] 日期：2009.4.10

第一章 引 言

1.1 TCP 信头压缩的发展

过去十年中，通信技术、互联网和多媒体技术的综合完全改变了通信领域，在此期间，通信业的重点在于建立新的网络，用于提供更广泛的音频、数据和视频传输，提供独立于用户位置的各种服务。第三代无线通信系统（IMT-2000）将提供更重要的应用：聚合了音频、数据和视频的真实移动服务[1]。

通信系统使用的无线频带是有限而且是十分宝贵的，容易受到带宽、时延和误码率的影响，对于运营商是最具有价值的资源，应该优化使用这些频带。因此无线系统设计中最重要的方面是如何选择网络平台，获得预期的 QoS 保证，以便和固网链路竞争。已经可以预见，未来的集成系统将采用全 IP 的网络平台，需要全部和有效地支持 TCP/IP 协议[2]，这不仅用于有线链路，而且用于无线链路。

低带宽 IP 协议的有关性能问题从 1984 年开始在细线（Thin-wire）协议上进行研究[3]。1990 年 2 月，Van Jacobson 提出了以信头冗余信息为基础的低速串行链路上压缩 TCP/IP 信头（CTCP，Compressing TCP/IP Header for Low-Speed Serial Links）的机制[4]，可以将 40 个字节的 TCP/IP 信头压缩到 3~6 个字节。CTCP 机制采用的是计时超时的差错恢复机制，不适用于往返时间较长的无线链路。瑞典 Lulea 大学的 Dr.Stephen 和 Dr.Mikael 等人在 1999 年 2 月开发了 IP 信头压缩协议（IPHC，IP Header Compression）[5]，IPHC 压缩方案对 CTCP 提供了一些扩充，最重要的是提供了对 UDP、IPv6 和额外 TCP 特性的支持。这个协议在点到点的链路上可以压缩每跳上的多个 IP 信头、TCP 信头和 UDP 信头，可以用于无线链路的分组传输环境。但是，对于高误码率（BER）、高丢失率、长往返时延（RTT）的无线链路上，并不具有强的鲁棒性，影响了数据传输的质量。S.Casner 和 V.Jacobson 共同开发了低速链路上压缩 IP/UDP/RTP 信头的协议(CRTP，Compressing IP/UDP/RTP Header for Low-Speed Serial Links)[6]，它能解决在低速串行链路上传输语音和视频报文时遇到的一些问题。CRTP 主要对 RTP/UDP/IP 信头进行压缩，适用于往返时间短的本地链路，但在往返时间长的无线链路（如蜂窝链路）上压缩性能并不好，因此也不适用于无线链路。

通过研究表明，这些协议在无线链路上的性能比较差。信头压缩的下一步目标是在高误码率、低带宽的无线链路上增加鲁棒性和压缩效率。文献[4]提议的算法用来提高 TCP/IP 流的性能和鲁棒性，它提供了一种可以从丢失压缩同步中进行恢复的策略，这种算法进一步降低了同步丢失的概率。文献[7]作了进一步的改进，它提议了两种传输策略：含有静态压缩启动的自适应提议（ABP-SCS，Adaptive Base Proposal with Static Compression Start）和含有动态压缩启动的自适应提议（ABP-DCS，Adaptive Base Proposal with Dynamic Compression Start），可以降低 UMTS 链路上 TCP/IP 同步信头的丢失率。文献[8]中提议的 TCP 检测鲁棒性信头压缩（TAROC，TCP Aware RObust header Compression）方法，以及文献[9]中提议的高效金字塔编码算法（EPIC，Efficient Pyramid Image Coder）方法对 TCP/IP 信头的压缩有了进一步的贡献。TAROC 的目标是使用 TCP 拥塞窗口机制限制错误的扩散，EPIC 使用变化的哈夫曼编码来提供一系列的压缩信头方案。

为了能在无线链路上获得较高的压缩率和较好的抗差错鲁棒性，需要开发一种可行的鲁棒性信头压缩方案。IETF 工作组于 2001 年提出的鲁棒性信头压缩（ROHC，RObust Header Compression）[10]，能在无线链路上获得较好的性能，在 2007 年 2 月又进行了修正和补充[11]。ROHC 适用于误码率高，往返时间较长的无线链路，可对 RTP/UDP/IP、UDP/IP、TCP/IP 等多种信头进行压缩。作为鲁棒性和可扩展的方案，ROHC 代表了当前信头压缩的技术水平。

文献[12]针对 IP 无线网络内 TCP 信息流提出了一种新的信头压缩方案，实现了无线全 IP 网络中 TCP/IPv4 数据流的信头压缩，并对其性能进行了分析，为以后针对 IPv6 的信头压缩方案及性能分析提供了参考。

但是 ROHC 无法高效压缩 TCP 流的信头，ROHC+[13]在 ROHC 的基础上提出了一种能有效压缩 TCP 流信头的简档和算法。为了减少差错传播，ROHC+采用 W-LSB 编码，并结合使用三种不同的修复机制处理受损的压缩分组。ROHC+还针对 ROHC-TCP 压缩，提出了文景复制机制[14]。

RFC4995[15]在 RFC3095 的基础上进一步定义了 ROHC 框架的相关内容，包括符合 RFC4163[16]需求的 TCP 简档。RFC4996[17]提出了一个 TCP 信头压缩方案，增强了鲁棒性和兼容性，尤其是针对含有 TCP 选项信头域的压缩。其中使用了 RFC4995 中提出的 ROHC 协议框架，支持 RFC4164[18]中定义的上下文复制策略，对于 RFC4413[19]中的短暂 TCP 数据流，上下文复制是非常有用的。

2008 年 4 月，IETF 工作组正式推出了 ROHC 压缩的改进版本 ROHCv2[20]。在同样的操作条件下，RFC5225 和 RFC3095 可以获得相同的压缩率和鲁棒性，它们最小压缩信头的大小和比特布局是一致的。但 ROHCv2 简化了规则和算法，增强了分组丢失或重排的鲁棒性机制，而且可以处理任意数量的 IP 信头。

1.2 信头压缩的基本原理

在无线链路上运行 IP 协议非常困难，最主要的原因是这些协议在封装和分割过程中产生较大信头开销。众所周知，每层的 TCP/IP 架构在每个分组中都引入控制信息（信头），应用程序的带宽被数据和控制信息所分享。例如 IPv6 信头本身为 60 个字节，对于移动 IPv6 可能达到 100 个字节。分组中负载越小，信头所占的比例越大，产生的影响也就越明显，个别情况下，某些应用程序产生的负载大小接近或小于信头大小。因此，对于负载比较小的典型应用，无线环境中引入的信头带来明显的延迟，使 TCP/IP 平台上的服务难以实现。另外，某些 TCP/IP 协议栈在无线环境中产生特别的问题，如 TCP 的拥塞控制和重传机制，这些特别的问题限制了 TCP 协议的性能，降低了高速数据吞吐量。

为了避免浪费无线信道带宽来传输控制信息，要引入信头压缩机制。在同一个分组内，特别是在同一个流的连续分组中，信头中存在明显的冗余。如 TCP/IPv6 信头中，IPv6 的版本是已知的，源/目的地址、源/目的端口和下一信头等字段都是固定不变的，IPv6 信头中的有效载荷长度字段、UDP 信头中的数据报长度字段是可以推断出来的。信头压缩的基本原理在于：在数据流持续期间，仅传输有限次数的全部信头信息，后续的分组中只传输发生变化的信头字段及同一分组流的关联标识符，未传送字段可以根据依赖性和可推断性，重新计算产生，就可以达到信头压缩的目的。

1.3 目前存在问题

虽然 ROHC 的应用已经比较成熟，但 ROHC 还有很多不完善的地方。ROHC 的运行参数都是静态设置的，针对具体业务和使用的信道类型采用不同的参数优化配置[21]，这些参数值一般都是固定设置的，当链路状态发生变化时，不能随时进行调整。文献[22]具体研究了这些参数的性能配置，推荐了一些优化可行的配置参数。而实际应用中，无线链路容易受到外界的影响，传输质量可能随时发生变化。ROHC 信头压缩方案在设计时没有将无线信道的变化状态考虑在内，当无线

信道进入不同状态时，采用的压缩方法一成不变[23]。由于压缩方案无法根据不同的信道状态而采取灵活的措施，因此无法捕捉不断变化的无线信道链路特性并随之进行优化，导致了较低的工作效率。

现存的TCP/IP压缩方案没有压缩信头的握手信息（SYN和FIN），因为许多短期TCP连接占用了信道，对这些握手分组的压缩可以有效提高整个信头的压缩效率[17]，然而在以太网的链路上，对于较小的分组进行压缩填充时会出现问题[24]。类似的是，在处理TCP选项字段上也存在局限性[25]，选项字段的任何改变（例如时间戳和SACK），都会使整个字段不能压缩[26]，从而大大降低了TCP的压缩效率。

基于ROHC的工作原理，根据上下文的文景进行解压，这就要求分组必须按顺序到达，不适合两个端点间存在多条到达链路的情况。另外，在分组压缩和解压处理上，由于ROHC的工作机制比较复杂，在实现和控制上都需要消耗较多的系统资源。

1.4 主要研究工作

在本文中，研究了ROHC信头压缩的算法和工作机制，分析了TCP/IP信头压缩的局限性，针对无线IPv6网络中TCP/IP信头压缩的技术和性能，改进了ROHC和TCP压缩方案不完善的地方，在原型系统上进行了实现，通过仿真测试验证了结果。

本文的主要工作如下所述：参照ROHC分类标准，给出了TCP/IP信头字段的分类方式，按照ROHC简档规范，采用W-LSB编码方法，编写了TCP/IP分组的压缩简档。在无线链路中，信道质量可能快速变化，很难找到适合各种情况的优化的滑动窗口宽度值，需要采用灵活的可调节滑动窗口宽度机制。提出了滑动窗口宽度的动态调整建议，该方法没有改变压缩和解压状态变迁逻辑，也没有引入附加的控制信息，兼顾了强鲁棒性和低系统开销。为了方便ROHC简档的编写和调试，在ROHC压缩器端增加处理函数，可以产生适用于任何类型的测试分组，随时检验简档的压缩和解压缩功能，特别适用于面向连接的TCP/IP流。

1.5 论文结构

本文共分六章，各章内容如下：

第一章 回顾了TCP信头压缩发展的历程，介绍了冗余信头压缩的基本原理，分析了目前ROHC

不完善的地方及TCP/IP压缩方面的缺陷。

第二章 概述了ROHC协议的基本原理、工作状态和运行模式，详细描述了协议中的主要压缩算法W-LSB，介绍了ROHC压缩和解压缩的实现流程，在此基础上给出了ROHC在Linux中的接口实现。

第三章 编写了TCP/IP压缩简档。按照ROHC简档规范和TCP信头字段的变化方式，引入MSN关键字段，设计了TCP/IP分组的压缩简档格式，并得到实现。

第四章 分析了滑动窗口宽度动态调整的需求和对性能的影响，提出了滑动窗口宽度动态调整的方法，兼顾了强鲁棒性和低系统开销，并通过仿真进行了测试和验证。

第五章 为了方便编写和调试TCP/IP压缩简档，给出了ROHC系统测试信源的实现方式，介绍了测试信源的使用，展示了测试信源不同于其它业务信源的优点；描述了用户接口和统计功能的实现。

第六章 对全文总结，描述整个论文的研究历程和取得成果，并对下一阶段的研究和发展进行了展望。

第二章 ROHC 协议及其实现

2.1 ROHC 的基本原理

为了实现信头压缩，需要在 TCP/IP 架构中加入两个新的功能单元：信头压缩器和信头解压器，如图 2-1 所示。

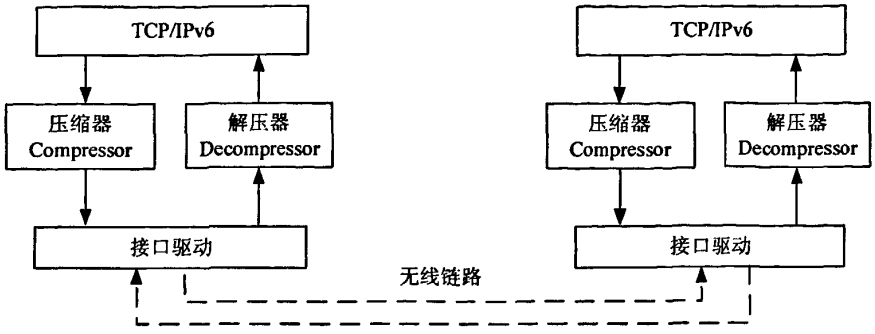


图 2-1 TCP 压缩基本框架

ROHC 系统包含压缩和解压两个部分，压缩部分功能是对 IP 分组进行压缩处理，形成 ROHC 压缩分组，解压部分功能是对压缩后的 ROHC 分组进行解压处理，还原成原来的 IP 分组。

压缩方在刚开始发送 IP 分组时，ROHC 协议首先创建一个对应于此数据流的压缩器，并将这些分组信息（也称为文景，Context）和关联标识符保存下来，然后将包含完整信息的 ROHC 分组通过无线链路发送到解压方。解压方根据此 ROHC 分组创建一个对应于此数据流的解压器，解压器接收到 ROHC 分组时，也首先将这些信息保存下来，解压后将还原的 IP 分组交给 IP 协议处理。对于同一数据流中的后续 IP 分组，压缩器参照保存的信头信息，将分组中不变的冗余信息去掉，仅包含变化的信头域和关联标识，形成 ROHC 压缩分组，传递到解压方。解压器根据先前保存的完整信息和关联标识符对 ROHC 压缩分组解压，形成 IP 分组并交付给上层 IP 协议处理。

2.2 ROHC 的工作状态

ROHC 压缩器有三种压缩状态，每一种都是不同程度的压缩，分别为：IR 状态（Initialization and Refresh）、FO 状态（First Order）和 SO 状态（Second Order），这三种压缩状态级别依次升高，压缩器总是从最低压缩状态 IR 状态开始工作，然后依次向高级压缩状态转移。每个工作模式下每种压缩状态都使用不同的信头类型。

IR 状态发送所有信头信息，刚开始工作时，无压缩文景可用，IR 状态的作用是初始化

解压器端的静态文景；正常工作以后，IR 状态用于解压失败后修复文景。在 IR 状态下，压缩器发送全部信头信息，包括静态字段和动态字段以及附加信息，全部以非压缩方式传送。当解压器已经收到足够的信息并完成对文景的初始化后，压缩器才向高级压缩状态转移。

FO 状态用于有效地交流信头中的动态部分。当压缩器确认解压器已经获得文景的静态部分，就进入 FO 状态，只对动态字段的可变部分（和上一个分组相比，信头中改变的部分）进行编码传送。在此状态下，信头被部分压缩，静态部分仅有少量被更新。

SO 状态下压缩率最高，当数据流“规范”（指在已知关键字段的情况下，所有动态字段的改变可以预期）时，在 SO 状态下仅传送关键字段的编码部分，信头大小可以非常小（TCP/IP 压缩可以到 3～5 个字节，RTP/UDP/IP 可以到 1～3 个字节）。此时压缩器仅向解压器发送经过压缩的 SN 和文景标识符（CID，Context ID）等一些必要的信息，解压器主要是对被压缩的 SN 进行解压，其它所有字段的值都可从文景中获得。压缩器大部分时间处于 SO 状态，当信头不再符合规范的格式时，也就不能利用当前的文景信息进行压缩，此时压缩器就会离开这个状态，向下转移到 FO 或 IR 状态。

对于 TCP 数据流，由于采用了文景复制机制，压缩状态变化略有不同。压缩器工作在 IR（含 IR 和 IR-CR）和 CO 两种工作状态，IR 是初始化状态，IR-CR 是初始化和文景复制状态，CO 是最高压缩状态，其压缩状态机如图 2-2 所示。

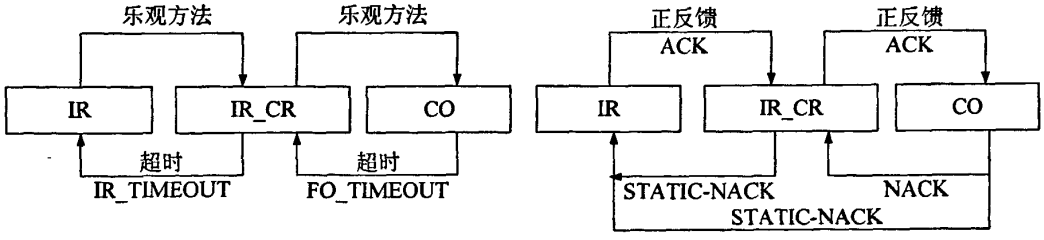


图 2-2 TCP 流的压缩状态机

对于工作在 O 模式下的 TCP 流，有两种转移方式：一是没有反馈信道时使用乐观方法；二是在反馈信道存在时使用反馈方法。对于 O 模式下没有反馈信道的 TCP 流，压缩器在发送足够分组，确认解压器接收到了正确的静态信息后，从 IR 状态向 IR_CR 状态转移，同样在 IR_CR 状态发送了足够的分组后向 CO 状态转移。向下转移采用周期性原则，当超时机制 FO_TIMEOUT 或 IR_TIMEOUT 达到时，压缩器向下转移到低级压缩状态（参见图 2-2 左边部分）。对于使用反馈信道的 O、R 工作模式，初始状态处于 IR 状态，当接收到正反馈（ACK 分组）时转移到 IR_CR 状态，在 IR_CR 状态收到正反馈后转移到 CO 状态。当压缩器在高级压缩状态接收到负反馈（NACK 或 STATIC-NACK）时向下转移到低级压

缩状态（参见图 2-2 右边部分）。

CO 状态目的是有效传输分组流中变化的信息，保持最高压缩效率。压缩器在 CO 状态接收到 STATIC-NACK 反馈分组时，转移到 IR 状态，这时压缩器应该重新选择基本文景。如果选中基本文景，则压缩状态转移到 IR-CR 状态，向解压方发送 IR-CR 分组，解压器对解压文景进行初始化和刷新操作。如果没有合适的基本文景，则压缩器停留在 IR 状态，一直向解压方发送 IR 分组。

ROHC 解压器也有三种状态：无文景状态（No Context，NC），静态文景状态（Static Context，SC），全文景状态（Full Context，FC），其解压状态机如图 2-3 所示。

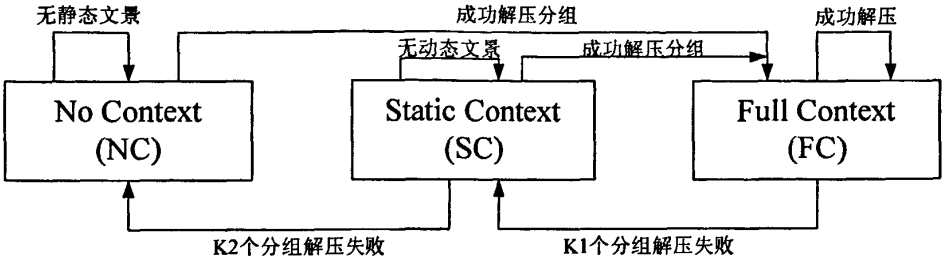


图 2-3 TCP 流的解压状态机

初始时解压器还没有成功解压一个分组，只能工作在最低解压状态：NC 状态。NC 状态主要是一个新数据流刚开始解压处理时解压器所处的状态，NC 状态时，既没有 IP 信头的静态信息也没有动态信息，需要压缩器工作在 IR 状态发送包含完整信头的分组。解压器在 NC 状态只能解压 IR 分组，在接收到 IR 分组后，解压器使用 CRC 校验分组的正确性，校验正确后可以更新文景。当存在反馈信道时，解压器解压分组成功后将发送 ACK 反馈分组，如果校验失败，解压器发送 NACK 分组。在接收到 IR 分组并成功解压以前，解压器将丢弃所有接收到的 ROHC 压缩分组。

SC 解压状态指解压器获得了足够的静态文景信息时所处的状态，希望接收到包含完整动态信头的 ROHC 压缩分组。

解压器只要成功解压了一个 IR 分组，就可以转移到 FC 状态。FC 状态指解压器获得了足够的静态文景和动态文景的变化规律信息时所处的状态，同压缩器的 SO 状态相对应，能够接收压缩器在 SO 状态所发送的 ROHC 压缩分组。

在 FC 状态下，当最近收到 N1 个连续的分组中有 K1 个解压失败时，解压器就会转移到 SC 状态。在 SC 状态下，当成功解压一个含有足够更新信息的分组时，解压器就会返回 FC 状态。如果在 SC 状态下，当最近接收到 N2 个连续的分组中有 K2 个解压失败时，解压器就转移到 NC 状态。

2.3 ROHC 的工作模式

ROHC 定义了三种工作模式：单向模式（Unidirectional，U-Mode），双向乐观模式（Bidirectional Optimistic，O-Mode），双向可靠模式（Bidirectional Reliable，R-Mode）。每种模式下都有不同的压缩状态和解压状态，不同的工作模式对应的压缩/解压缩状态转移方式也不同。

（1）U 模式（U-Mode）：当不存在或不能使用反馈信道时，ROHC 工作在 U 模式下，此时分组只能向一个方向发送，即从压缩方向解压方发送。压缩器和解压器工作在 U 模式时，向高级状态转移采用乐观逼近原则。乐观逼近原则是指压缩器在 IR 状态连续发送 n1 个分组或者 FO 状态连续发送 n2 个分组后，会认为解压器已经收到了成功解压所需的相关信息，就会由当前状态向更高状态转移。由于无线链路的误码率较高，往返时延较长，随着压缩的进行，可能会出现压缩文景和解压文景不同步的情况，由于 U 模式没有反馈，压缩器无法得知文景是否同步。为防止这种情况的发生，压缩器在高压缩状态工作一段时间后，按照周期性原则转移到较低压缩状态。周期性原则是指当压缩器工作在高级压缩状态达到一定的时间 TIMEOUT 时，压缩器要向低级压缩状态转移，这样解压文景会被及时地更新，保证了两端文景的同步。U 模式下的压缩状态转移如图 2-4 所示。

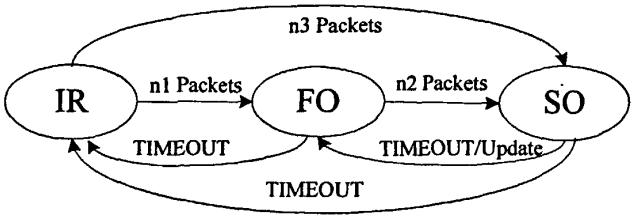


图 2-4 U 模式下压缩状态转移图

（2）O 模式（O-Mode）：O 模式和 U 模式较为类似，向上转移采用乐观逼近原则，周期性发送足够的分组，确信接收端已经建立了正确的解压文景，然后向高级压缩状态转移。但 O 模式下的反馈信道是可选的，向下转移不采用周期性原则，而是根据收到解压方的负反馈（NACK、STATIC_NACK 分组）向下转移。O 模式的目的在于最大压缩率和使用空闲反馈信道，O 模式下的压缩状态转移如图 2-5 所示。

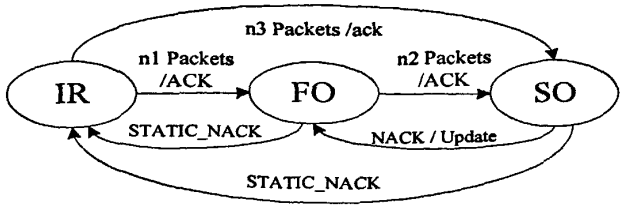


图 2-5 O 模式下压缩状态转移图

（3）R 模式（R-Mode）：R 模式与前两种模式不同，可靠链路中 R 模式使用反馈信道比较频繁，采用严格的状态转移逻辑控制，以防压缩文景和解压文景失去同步。压缩状态的转移依赖于解压方的反馈，解压缩成功后解压方发送正反馈，压缩器向高级压缩状态转移，解压失败后解压方发送负反馈，压缩器向低级压缩状态转移，如图 2-6 所示。

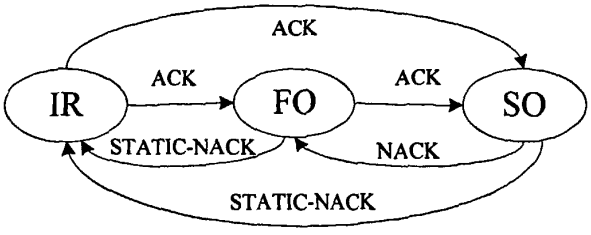


图 2-6 R 模式下压缩状态转移图

三种模式相比，由于 U 模式没有反馈机制，在较差的信道中，U 模式性能不如 O 和 R 模式，R 模式的鲁棒性最好，但是由于 R 模式比前两种模式使用的反馈都多，导致反馈占用一部分带宽。相比 U 模式和 R 模式，O 模式带来的开销比较大。U 模式适用于对实时业务产生的信头进行压缩，为保证压缩的鲁棒性，当压缩器工作在 U 模式下时，所有发送到解压端的压缩信头必须携带一个 CRC 校验和。

三种模式之间可相互转移，模式转移通常由解压器发起，解压器根据解压情况、无线信道的状况（例如反馈能力、错误概率和干扰）等因素决定是否要发起模式转移。当无线信道质量比较好时，ROHC 可以工作在 R 模式或 O 模式；如果无线信道质量很差且解压器不能向压缩方发送反馈的时候，ROHC 一般要工作在 U 模式。

2.4 ROHC 的压缩算法

最低有效位（LSB，Least Significant Bit）和基于窗口的最低有效位（W-LSB，Windows-Least Significant Bits）编码算法是 ROHC 协议中非常重要的压缩编码算法，贯穿于 ROHC 的整个压缩和解压过程，一般用于对连续分组中数值变化不大的字段进行压缩编码，要压缩的字段与已正确传递并作为参考的字段相比较，只传递二者不同的最低位比特，起到压缩的效果。解压器根据解压文景中保存的对应字段的参考值，对接收到的 LSB 编码或 W-LSB 编码的字段进行解压，将这些字段恢复到编码前的原始值。

2.4.1 LSB 算法

LSB 为最低有效位编码算法，利用 LSB 编码方法，对连续分组中变化不大的字段值进行编码，压缩器只传输字段值的最低 k 个有效位，而不是原始字段值。k 是一个正整数，

是要压缩的值 v 与已正确传递而作为参考值的 v_ref 二进制编码相异的最低有效位个数。解压器接收到 k 个比特之后，使用先前正确接收到的值作为参考进行解压，获得原始值。

现在举一个简单的例子来对此进行说明：假设要传送数 15，原来有一个成功的传递数 10 作为参考。按字节大小，10 的二进制值为 00001010，15 为 00001111，我们可以看到只有最低的三位不同，前边的都一样，于是 k 可以取 3，在传递时只传递最低的三位比特，就是 111。解压器接收到 111 后，用 111 来取代解压器参考值 10 的最低三位，这样便能够得出解压值为 15。

LSB 方法求 k 值的公式为：k=g(v-ref, v)= Length(v_ref ∧ v)，其中∧为异或运算符，Length 为取二进制编码长度值的函数，函数 g 是在 LSB 算法中求解两个数二进制编码不同的最低有效位数。

LSB 方法虽然简单，但每个信头的解压需要上一个信头的正确接收。这种方法不适合于无线链路，因为单个分组丢失时，用于下个分组解压的文景无效，导致丢失压缩/解压缩文景同步，于是下一个分组不能正确解压而被丢弃。依此类推，所有后续的解压文景都无效，引起分组丢弃的雪崩现象，直到高层（例如 TCP）触发重传机制，这种现象称为损害扩散（Damage propagation）。

2.4.2 W-LSB 算法

文献[7]又提出了 W-LSB 编码算法，它具有更好的鲁棒性。W-LSB 编码算法使用含有多个参考值的滑动窗口，传输过程中即使某个参考值丢失，解压器只要接收到其它参考值中的任意一个，就可以正确解压 LSB 编码的值，图 2-7 举例展示了 W-LSB 压缩算法。

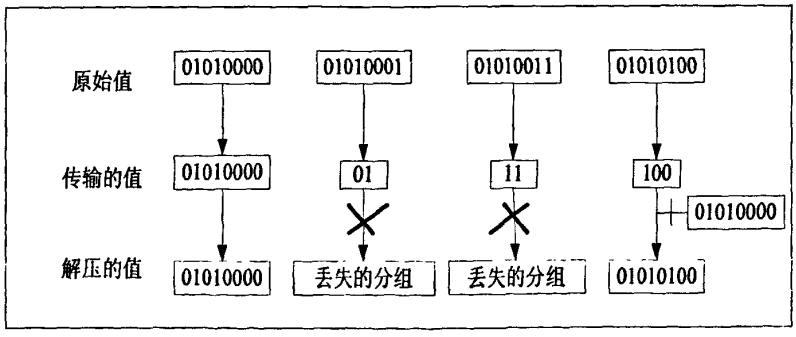


图2-7 W-LSB压缩算法举例

W-LSB 压缩算法：

（1）压缩端每发送一个经过 CRC 验证的字段值 v（成功或未成功压缩）后，压缩器就把 v 添加到发送端的滑动窗口内。

（2）最低有效位 LSB 求解公式：LSB= v & [(1<<k)-1]，其中，&是按位与运算符，<<是左移运算。

（3）为保证解码准确无误，W-LSB 编码、解码的取值区间必须保留原始值的信息，并保证在此区间上，原始值是唯一一个与这 k 个比特相对应的域值，这个区间就是译码间隔 f(v-ref, k)。f(v-ref, k)求解函数为：f(v-ref, k)=[v_ref-p, v_ref-(v_ref & (2^k-1))+ (2^k-1)-p]，其中，v_ref 为压缩 v 时从滑动窗口中选取的参考值，&为按位与运算符，参数 p 用于使该取值区间可以针对域值的特点进行适当的平移，从而使编码方法更加有效。对那些总是增长的域，p 可置为-1。

（4）W-LSB 方法求 k 值的公式为：k = max(g(v_min, v), g(v_max, v))，其中，k 为传递压缩值 v 的最低有效位数；v_min 是滑动窗口中的最小参考值，v_max 是滑动窗口中的最大参考值。用滑动窗口的最大和最小值作为参考值，计算出 W-LSB 的比特位个数，并取最大的位数作为 k 值。

（5）当压缩器或解压器确信某个参考值 v_ref 不再可能用作参考时，窗口中可以删除这个参考值 v_ref，窗口进行滑动。在 R 模式下，如果接收到解压器发送的 ACK 分组，则意味着压缩器可以从滑动窗口中删除小于确认值的旧参考值。

（6）解压器接收到最低有效位 LSB 后，通过下面的解码算法公式获得解码值 v：v=[(v_ref>>k)<<k] | LSBs，其中，解码参考值 v_ref 来自已经成功解压的值，|是或运算符。

ROHC 压缩和解压中 W-LSB 算法流程如图 2-8 所示。

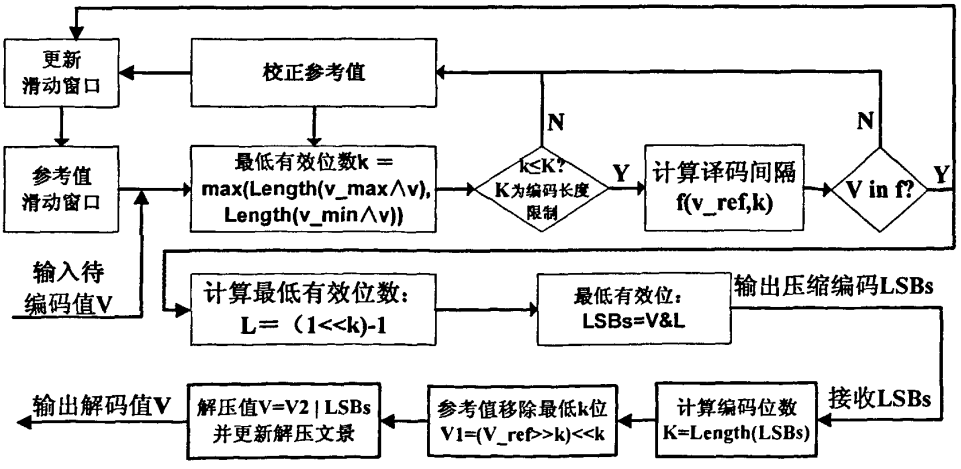


图 2-8 W-LSB 算法流程图

W-LSB 算法中滑动窗口宽度（SWW，Slide Window Width）是一个影响 ROHC 压缩率和鲁棒性的重要因素。W-LSB 编码的鲁棒性在于如果连续分组丢失个数不超过 SWW 窗口宽度，即丢失分组的个数小于参考值的个数，解压器仍然可以正确解压。如果窗口宽度

SWW 过小，用做压缩的参考值就少，需要压缩的比特位数 k 短，可以提高压缩效率。但是一旦在无线链路上丢失 SWW 个以上的分组，超出了参考值可以译码的范围，解压方就不能够正确解压，影响了 ROHC 的鲁棒性。相反，如果 SWW 取值过大，保存的参考值较多，虽然能够保证 ROHC 的鲁棒性，但是由于要压缩传输的比特位数 k 变长，从而降低了压缩性能。在第四章中将介绍 SWW 宽度动态改进的方法。

2.5 ROHC 压缩的实现

当网络层发送 IP 分组时，首先进入 ROHC 压缩处理模块，处理过程如下，具体流程如图 2-9 所示：

（1）判断网络设备是否允许进行 ROHC 压缩。如果此设备不允许压缩，就将该分组直接发送出去；如果此网络设备允许压缩，接着判断分组的类型，看是否为 IPv6 分组，如果不是 IPv6 分组，不进行压缩，直接把此分组发送出去；如果是则进行下一步操作。

（2）确定分组简档类型。分析此 IP 分组的信头，依次处理分组的下一头域，直到有效载荷部分或者结束（网络控制报文可能没有有效载荷数据），确定压缩的简档类型。

分析过程为：根据 IPv6 的下一头域逐次进行处理，如果存在 ESP 扩展信头，则此种类型的分组采用 ESP 简档；如果没有，接着分析网络层的上层协议信头，如果为 TCP 协议信头，压缩时采用 TCP 简档；如果是 UDP 协议且上一层为 RTP 协议信头，则采用 RTP 简档；如果分组存在 UDP 协议信头而上层不是 RTP 协议信头，采用 UDP 简档进行压缩；最后，如果分组只有 IPv6 协议信头和网络控制报文协议信头，则采用 IP 简档进行压缩（IPv4 分组不进行压缩处理，第五章中就是利用了 ICMP 信头来产生测试信源的）。

（3）确定数据流。区分数据流，确定所使用的压缩器，如果此压缩器还没有创建，就要创建一个与此数据流相关的压缩器。

（4）如果压缩器是新建的，就要根据分组的简档类型，初始化此压缩器的文景；如果压缩器已经存在，则比较 IP 分组信头与对应简档文景的相应域，发现变化的部分，确定要发送的信头域。

（5）根据压缩器的工作状态和模式以及要发送的信头域，确定发送 ROHC 分组的类型。在 IR 状态，压缩器要发送 IR 类型的分组；在 FO 状态可以发送 IR_DYN 和 UOR-2 类型分组；在 SO 状态可以发送 R-0、UO-0、R-1、UO-1、R-0-CRC 类型分组。

（6）确定了分组类型，调用具体压缩简档，对这种分组类型的各个字段进行编码压缩，生成 ROHC 分组。注意：不同类型 ROHC 分组需要调用特定压缩简档进行编码。

（7）当压缩器调用特定压缩简档对 ROHC 分组进行正确的编码后，压缩器需要进行

一些状态和统计信息的处理，如果需要进行状态转移，就进行状态变迁。

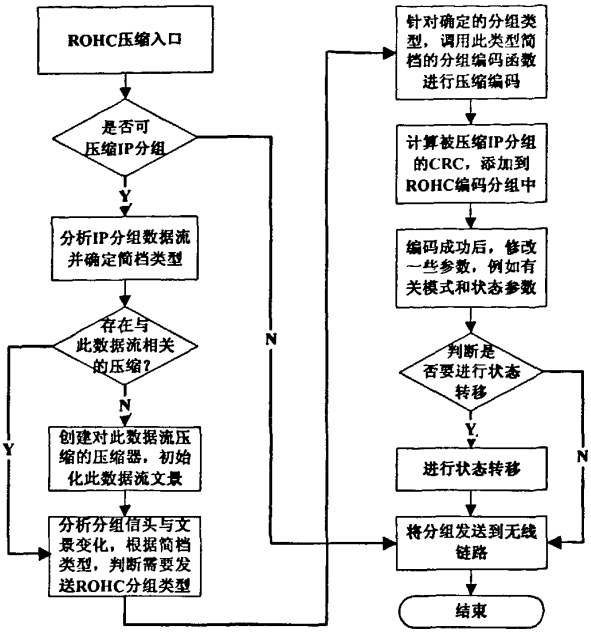


图 2-9 ROHC 的压缩流程示意

（8）压缩器将 ROHC 分组发送出去，整个压缩处理过程结束。

2.6 ROHC 解压缩实现

当 ROHC 压缩分组进入解压器后，解压操作步骤如下，具体流程如图 2-10 所示：

（1）首先判断接收 ROHC 分组的类型。如果不是 ROHC 反馈分组，跳到第（3）步执行；如果是反馈分组，说明解压器接收到一个来自其它网元解压方的反馈分组。

（2）当接收到一个反馈类型的分组时，根据反馈分组中的文景标识符，确定所反馈的压缩器，对此压缩器进行处理。解压器会分析反馈分组的类型，如果为类型 1 反馈，就说明是一个正反馈，调用压缩器的状态转移函数进行状态转移；如果为类型 2 反馈，判断反馈分组中的模式域值与当前压缩器的工作模式是否一致，如果不一致，就要调用压缩器的模式转移函数进行模式转移；如果与当前压缩器的工作模式一致，则进行状态的转移，本次 ROHC 分组接收处理过程结束。

（3）操作执行到这一步说明接收到的分组不是反馈分组，而是 ROHC 压缩分组。首先判断 ROHC 分组的类型，然后分析信头信息，取出压缩分组所使用的文景标识符和简档类型，确定与此文景标识符相对应的解压器，如果没有对应的解压器，就要建立一个解压器，并对此解压器进行初始化。根据此压缩分组的简档类型和分组类型，调用特定简档对此类型的分组进行解压。

（4）调用 CRC 校验函数对解压分组进行校验，如果通过 CRC 校验，则认为分组正确解压。如果发生错误，则利用错误修复机制进行修复，再进行解压。

分组成功解压后，根据被解压 ROHC 分组的更新文景特性对文景进行更新，更新参考值的滑动窗口，解压器要向高级解压状态转移，并且根据解压器的工作模式和被解压的 ROHC 分组类型发送正反馈分组。例如，如果压缩器和解压器工作在 R、O 模式，则成功解压 IR、IR-DYN、UOR-2 类型分组后，解压器就必须向压缩方发送正反馈分组，表示已经成功解压，压缩器可以向高级压缩状态转移。在信道情况很好的状况下，解压的成功率是很高的，可以在反馈分组中携带模式信息，让压缩器进行模式转换，解压成功率高的情况下，可以向高级模式转移，解压成功率低的情况下，可以向低级模式转移。

如果解压失败，则将该压缩分组丢弃，并判断解压器当前的工作模式和工作状态，如果是在 R、O 模式，就要向压缩器发送负反馈分组，表明解压器解压失败，要求压缩器向低级压缩状态转移，同时解压器向低级状态进行转移。

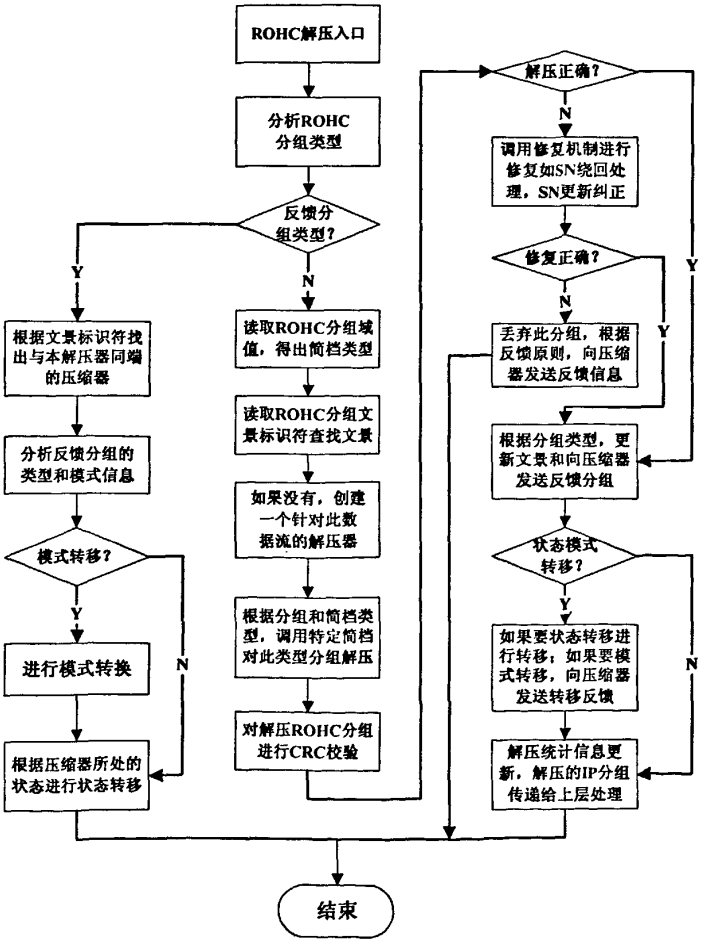


图 2-10 ROHC 的解压缩流程示意

（5）无论是成功解压还是失败解压，都要进行一些统计信息操作，同时解压器根据解压情况也要进行一些判断，看是否要进行模式转移，如果需要进行模式转移，反馈的分组携带模式转移的信息。

（6）正确解压的分组交付给上层 IP 协议进行处理，如果失败，则将分组丢弃，结束本次解压操作。

2.7 ROHC 在 Linux 中的实现

Linux 的网络接口分为四部分：网络设备接口部分、网络接口核心部分、网络协议部分以及网络接口 socket 层。网络设备接口部分主要负责从物理介质接收和发送数据，实现的文件在 linux/driver/net 下面。网络接口核心部分是整个网络接口的关键部分，它屏蔽了各种各样的物理介质，为网络协议提供统一的发送接口，同时负责把来自下层的分组往合适的协议派送，它的主要实现文件在 linux/net/core 下面，其中 linux/net/core/dev.c 为主要的管理文件。网络协议部分是各种网络协议的实现部分。Linux 支持 TCP/IP、IPX、X.25、AppleTalk 等协议，各种协议实现的源码在 linux/net 下有相应的名称，其中 TCP/IP 实现的源码在 linux/net/ipv6 下，linux/net/ipv6/af_inet.c 是主要的管理文件。网络接口 socket 层为用户提供了网络服务的编程接口，主要源码在 linux/net/socket.c 中。

2.7.1 Linux 内核网络接口分析

数据分组的发送过程：IP 层的数据分组首先进入发送队列 ip_queue_xmit，然后调用 dev_queue_xmit(skb)以套接字缓冲区 skb 的形式在网络设备上发送某一数据分组，网络设备由 skb->devname 指定。网络接口核心层通过 dev_queue_xmit()这个系统函数向上层 IP 层提供统一的发送数据接口，无论是 IP 协议还是 ARP 协议，都要经过这个函数把要发送的数据传递给网络接口核心层。函数 dev_queue_xmit()从 dev_base 指针所指的活动网卡设备链表中查找 skb->devname 注册的相应网络设备，然后调用网络设备的发送指针 devname→hard_start_xmit()来最终完成数据分组的发送工作，devname→hard_start_xmit 会调用实际的网络设备驱动程序来完成数据发送任务。

数据分组的接收过程：网卡正确接收完一个数据分组，会触发一项中断，中断处理例程假设为 net_interrupt（不同网卡驱动中这个函数的名字不一样）。一旦该中断被识别为一个输入数据分组，net_rx()就会承担进一步的处理，它调用 dev_alloc_skb()为这个数据分组分配一个套接字缓冲区结构体，用这个结构体对收到的数据分组进行封装，调用

eth_type_trans()识别出 2 层数据帧中的类型，最后调用 netif_rx()将数据分组放入输入队列中。调度器会检查当前系统中是否有激活的软件中断没有得到处理，有的话就通过 do_softirq 调度执行。这里最终会调用 net_rx_action()函数把收到的数据分组交给相应的协议处理函数来处理，如 IP 数据分组就交给 ip_rev()函数处理。

Linux 内核中数据链路层上数据分组的路径参见图 2-11。

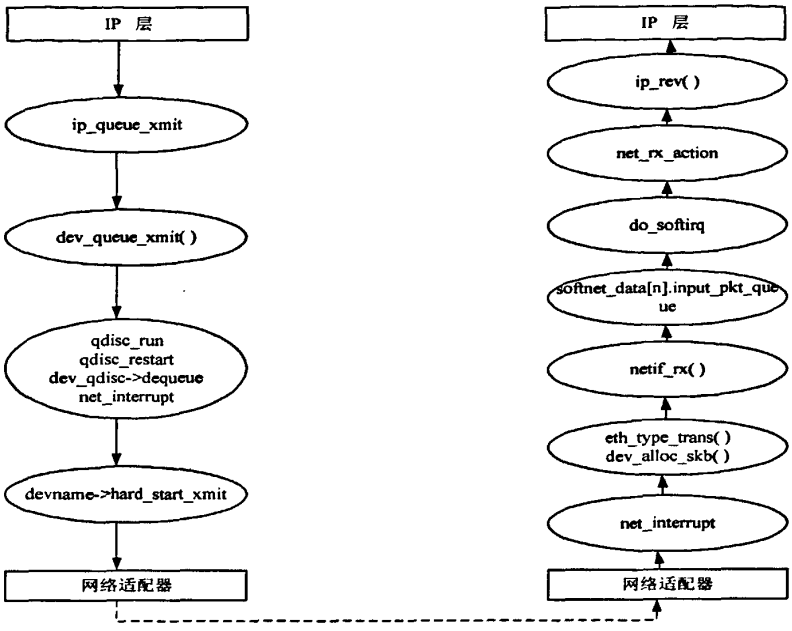


图 2-11 Linux 内核中数据分组收发流程

2.7.2 压缩数据时与 Linux 内核接口

对于发送，上层协议实体都会调用函数 dev_queue_xmit()来实现发送，最终的发送过程由 devname->hard_start_xmit()完成。Linux 将所有的外部设备看成文件，并用统一的逻辑设备和数据结构来管理外部设备及其驱动程序。在 Linux 内核中有一个全局指针 dev_base，dev_base 链表维持的是每个活动网络设备的逻辑结构，它指向节点类型为 net_device 的指针链，net_device 结构具有的功能如下：

```
struct net_device {
open();
stop();
hard _start_xmit ();
.....
}
```

它是对网络设备的一个抽象，所有的设备都有一个 net_device 结构，结构中的指针 hard_start_xmit()指向具体物理网卡驱动的发送函数。新设备要加入这一指针链可以通过 register_netdev(&net_device)来完成。上层协议实体想发送数据分组的时候，就可以顺着 dev_base 找到该设备 devname，然后调用 devname->hard _start_xmit()实行发送。

从上面的分析可知：由于最终的发送是通过函数 devname->hard_start_xmit()完成的，因此要嵌入 ROHC 压缩功能，可以采用钩子技术，把这个函数进行替换，使它指向 ROHC 协议的压缩处理函数。具体的替换步骤如下：通过函数 dev_get_by_name(eth0)找到网络设备（eth0 代表设备名称），该函数返回这个设备的 net_device 结构，将 hard_start_xmit()函数备份为 back_start_xmit()，然后把结构中的 hard_start_xmit()替换为 ROHC 的压缩函数 rohc_compress()。这样所有通过这一网络设备的数据分组都会调用函数 rohc_compress()对感兴趣的分组进行压缩处理，形成 ROHC 分组，然后再调用 back_start_xmit()进行发送（实际上是调用了原来的 hard_start_xmit()发送函数）。ROHC 协议对 IP 分组压缩成功并生成 ROHC 分组后，要将 ROHC 压缩分组中 MAC 信头协议类型的信头域定义为 ETH_P_ROHC（原型中数值定义为 0x8088）。对于不需要压缩的分组，直接调用 back_start_xmit()发送。这样就成功地实现了 ROHC 协议与 Linux 内核网络协议栈的压缩接口，参见图 2-12。

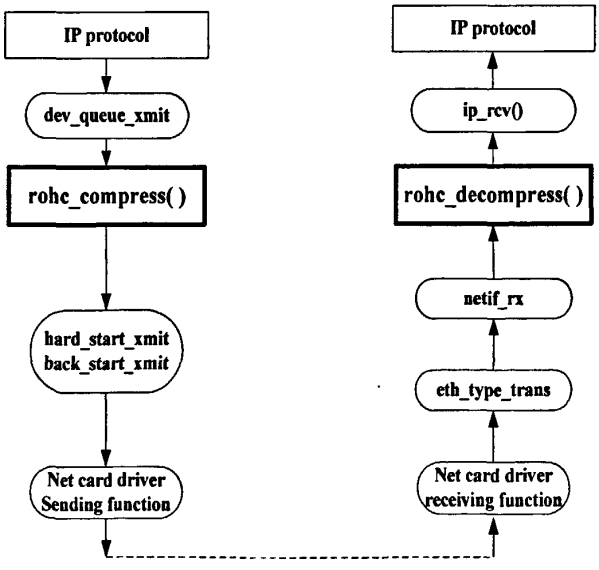


图 2-12 ROHC 与 Linux 网络协议栈接口示意

2.7.3 解压缩数据时与 Linux 内核接口

当网络设备接收到数据时，也是通过网络接口核心层与 IP 层的交互接口把分组传递到 IP 层的。内核中有一个全局数组 packet_ptype_base[]，这个数组分组含了可接收数据分

组的协议以及它们的处理函数。如接收 IPv6 分组时，调用 packet_ptype_base[ETH_P_IPV6]→func()函数（即 ipv6_rcv()接收函数），这样就把分组交给 IPv6 协议进行处理了。如果想加入新的协议，可以先声明一个结构体：

```
typedef struct packet_type{
.type = 协议名(如 ETH_P_IP);
.func= 接收函数(如 ip_rcv);
}
```

然后调用 dev_add_pack()（在 net/core/dev.c 中）就可以把该协议加入到数组 packet_ptype_base 中完成注册，从而 net_rx_action()就可以根据数据分组的类型到 packet_ptype_base 中找到相应的接收函数，交给其处理函数进行处理。删除是通过函数 dev_remove_pack()操作的。

具体操作步骤为：在 Linux 内核中，新建一个用于 ROHC 分组类型的 packet_type 结构，结构中包含了分组类型值（ETH_P_ROHC，定义值为 0x8088）和接收到 ROHC 分组的处理函数 func()（即 ROHC 解压函数入口 rohc_decompress()）。

这样，当 ROHC 分组到达解压方后，数据链路层通过 MAC 信头中的协议类型域值判断出此分组的网络层协议类型为 ETH_P_ROHC，并交付给 rohc_decompress()，即 ROHC 协议的解压函数处理。ROHC 解压函数对 ROHC 压缩分组进行解压，还原生成 IPv6 分组，再通过函数 netif_rx()调用 packet_ptype_base[ETH_P_IPV6]→func()函数，交付给 IPv6 协议进行处理。这样就成功地实现了 ROHC 协议与 Linux 内核网络协议栈的解压接口，参见图 2-12 所示。

2.8 本章小结

本章主要介绍了 ROHC 的基本原理、ROHC 压缩和解压缩状态变换、ROHC 的工作模式，详细阐述了在信头字段压缩上常用的 W-LSB 算法，给出了 ROHC 压缩接口和解压接口在 Linux 中的实现方式。ROHC 压缩策略保证了 ROHC 能够适应高误码率、低带宽、长时延的无线链路，在保证压缩率的基础上具有良好的鲁棒性。

第三章 TCP 的压缩简档

3.1 TCP 数据流

TCP/IP 是面向连接的流，在收、发两端间采用双向可靠的连接模式：这意味着数据交换前必须建立协议连接，一般是三次握手规则，之后 TCP 才能保证分发无错的分组到客户端。TCP 将应用服务产生的数据流进行分组，在发送的分组中，IP 层包含分组的负载并且序列号作为分组的第一个字节。在客户端，对于接收到的任何分组，TCP 发送含第一字节序列号的确认分组，表明客户端期望接收的下一个分组。在服务器端发送新的分组前，TCP 期望接收到所有发送分组的确认分组，如果期望超时，就认为分组丢失进行重发。TCP 连接可以被描述成两个流：数据流（服务器向客户端）和确认流（客户向服务器端），对这两种流采用分离和独立的压缩。

3.2 关键字段 MSN

每个 ROHC 压缩简档需要使用动态字段，称为关键字段，来获得最大的压缩比，关键字段可以选择相同流中连续分组间的固定变量。压缩器管理所有信头字段的变化，如果关键字段已知，其余字段变量全部可以推断，就进入 SO 状态，采用 W-LSB 编码，只发送关键字段的可变部分，在开销和鲁棒性上有较高的收益。在解压器端，如果接收到的压缩分组中仅有关键字段变化，采用对应的 W-LSB 解压算法，可以计算或预测其它字段的值，从而恢复整个原始分组。

对于 TCP 流的情况，具有较好规律性的动态信头字段根本不存在（TCP 序列号或确认序号的变化没有规律），因此引入一个附加字段，称为主序列号（MSN，Master Sequence Number）字段，两个字节长。主序列号 MSN 可以由信头中的现有字段建立，也可以由压缩器创建，这里选择由压缩器自主创建。对于每个新 TCP 流，压缩器将 MSN 设置为一个随机初始化值，传送时不进行压缩。字段 MSN 有两个功能：推断递增字段的值和区分发送反馈数据的分组。

MSN 在 IR 分组中完整传送，在 CO 信头格式中传送 W-LSB 编码信息。对于解压器，作为反馈信息的一部分经常发送 MSN 字段，压缩器可以通过 MSN 来判断解压器正处理哪一个分组。

3.3 TCP 字段的分类和管理

ROHC 协议针对 TCP/IPv6 信头中的各个字段变化情况，对它们进行了分类，便于对不同类型的字段采用不同的处理方式。根据 TCP/IPv6 信头的变化情况，将信头的各个字段划分为五种类型：

（1）静态型（STATIC）：在数据流传送阶段，字段的值是不变的。如 IPv6 信头中的版本号和下一信头字段。

（2）静态已知型（STATIC-KNOWN）：事先已知而不必发送的字段。

（3）静态定义型（STATIC-DEF）：静态定义型用于定义分组流的字段，便于和其它分组流进行区分，在整个数据流发送期间保持不变。IPv6 信头中的流标记、源 IP 地址、目的 IP 地址以及 TCP 信头中的源端口、目的端口字段均属于此类型。

（4）可推断型（INFERRED）：这种类型的字段值可由其它字段值推断得出。如 IPv6 信头中的数据长度、TCP 信头中的数据偏移字段。

（5）变化型（CHANGING）：这些字段的值是变化的，变化可能是有规律的，如 TCP 信头中的 SN 字段，也可能是无规律的，如 TCP 校验和。

TCP 信头字段的分类方法和建议的处理方式如表 3-1 所示：

字　段	大小（比特）	分　类	处理方式
Source port	16	静态定义型	仅在初始化分组中传递
Destination port	16	静态定义型	仅在初始化分组中传递
Sequence number	32	变化型	W-LSB 编码，随时更新
ACK number	32	变化型	W-LSB 编码，随时更新
Data offset	4	可推断型	不传送，解压端可计算产生
Reserved	4	变化型	初始时传送，随时更新
Flag U.A.P. R.S.F	6	变化型	初始时传送，随时更新
Window size	16	变化型	初始时传送，随时更新
TCP checksum	16	变化型	每次必须非压缩传送
Urgent pointer	16	变化型	每次必须非压缩传送

表 3-1 TCP 信头字段的分类和处理方式

ROHC 协议对不同类型的字段采用不同的处理方式。对于静态型和静态定义型字段，仅在初始时发送，压缩器和解压器获得了这些信息后，将这些信息保存在文景中，以后就不必再发送，解压器从解压文景中即可获知这些字段的值。静态已知型字段和可推断型字

段不必在ROHC分组中传送，但是对于可推断型字段，如果字段之间的推断关系发生了变化，则必须重新传送。对于变化型字段，若是无规律的变化（如TCP校验和），则在每个ROHC分组中都需要完全传送；对于变化有规律的字段只发送变化了的部分（如TCP协议中的序列号SN）。相邻分组间，如果没有发生分组丢失和乱序，SN的值会逐渐递增，这时只需发送变化了的最后几位比特即可。

在这些分类的基础上，为了管理TCP信头字段，表中的最后一列表明了采用何种信头压缩方式来优化压缩性能。需要注意到：

●TCP校验和与紧急指针是仅有的两个完全不规则动态字段，在连续的分组流中变化没有规律，因此不进行压缩，完全传送；

●TCP校验和存在于每一个TCP段内，因此必须存在于所有的压缩字段中；

●对于其它TCP动态字段（SN、ACK N和窗口大小），可以采用W-LSB编码来获得高效的信头压缩率；

●为了更有效地压缩，数据流和ACK流被分开压缩。实际上，TCP数据流的SN字段和ACK流的ACK N字段有相同的特性，类似于ROHC简档1中的RTP时间戳特性。

3.4 TCP压缩简档

在制定分组格式时，按照ROHC简档规范进行，这样设计的方案可以通用于整个ROHC架构。在每个格式中引入MSN关键字段，用于发送可变的格式。所设计的分组格式具有以下特点：

●IR和IR-DYN分组中的简档字段值设为9，用于区分特殊的TCP/IP流；

●信头静态域和信头动态域都是可变长度的，分别代表TCP信头中的静态域和动态域；

●设置Ex标志，Ex=1表示存在额外的字段，支持TCP窗口大小和TCP紧急指针的传输，也可用作扩大SN字段的传输范围；

●所有定义的格式均携带CID字段（0～16比特），MSN字段（4～16比特）和TCP校验和字段（2个字节）。因为这些字段在各个流的每个分组中都需要进行传递，这意味着最小的压缩分组是3个字节长度。

压缩器到解压器的分组主要有IR分组、IR-DYN分组以及ROHC压缩分组。

3.4.1 IR分组格式

IR分组包含原始协议信头中的所有字段，用于初始化文景或者刷新文景，IR分组只

能在 IR 状态下发送，如下图 3-1 所示：

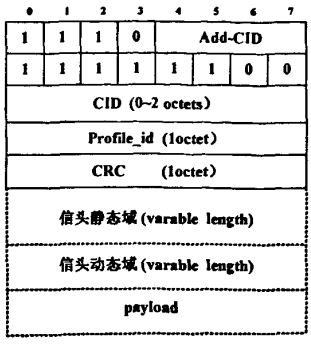


图 3-1 IR 分组格式

IR 分组携带原 IP 分组的全部信息（静态域和动态域）和关联标识符（PID、CID），并且还携带有 8bit 的 CRC 校验码，用于验证 IR 分组在解压方是否正确解压。其中，Add-CID 域为 4bit，其值为小号文景标识符（0～15）；第二个字节为分组类型标志，IR 类型的分组标志值为 11111100；CID 占 0～2 个字节，如果压缩方的 CID 取值范围大于 15（大号文景），则选用此字节，将 ADD-CID 设为 0，如果 ADD-CID 为非 0 值，则此字节不存在；Profile_id 为简档标识符，表示此 IR 类型 ROHC 分组使用的压缩简档类型；CRC 是 8 比特的校验值。

3.4.2 IR-DYN 分组格式

压缩器在 FO 状态时主要发送 IR-DYN 分组，此类型分组信头仅携带原 IP 的动态信头域，用于初始化或刷新文景中的动态部分。IR-DYN 分组格式如图 3-2 所示：

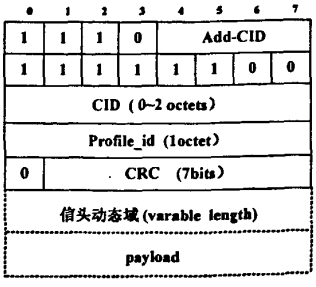


图 3-2 IR-DYN 分组格式

3.4.3 压缩分组格式

ROHC 压缩分组又分为三种类型：分组类型 0、分组类型 1 和分组类型 2。ROHC 压缩分组的表示形式为 Mode-Type-Some Property，如 R-0-CRC 分组，R 表示此分组可用于 R 模式，类型为 0，CRC 表示该分组含有 CRC 的变化信息。这样便于在不同的模式和状态

下采用不同的分组格式，提高了压缩效率。

分组类型 0 主要包括 R-0、UO-0 和 R-0-CRC 分组，此类型的分组长度最小，压缩器和解压器之间只传递经 W-LSB 编码后的 MSN。图 3-3 为类型 0 的分组格式：

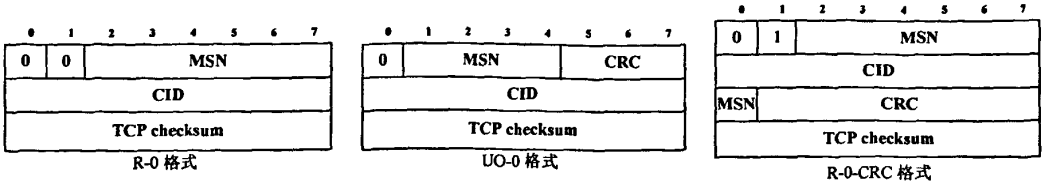


图 3-3 类型 0 的压缩分组格式

分组类型 1 主要有 R-1 和 UO-1 两种格式，用于更新关于 MSN 函数的参数。当 MSN 函数的相关参数改变或者所需的 MSN 位数超出了类型 0 的表示范围时，压缩器就会发送该类型的分组。图 3-4 是类型 1 的分组格式：

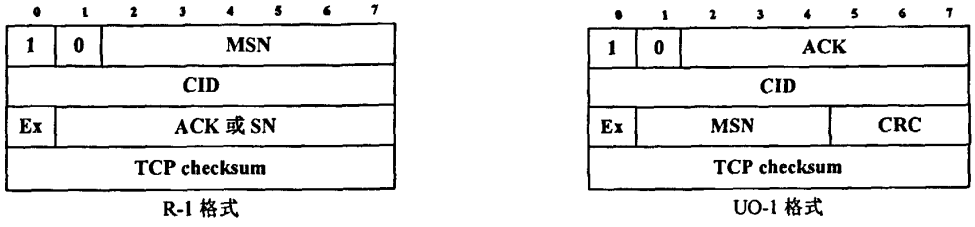


图 3-4 类型 1 的压缩分组格式

分组类型 2 主要是 UOR-2 分组，当某个动态域改变时，此分组用于更新动态域信息。类型 0 和类型 1 仅在压缩状态为 SO 时使用，类型 2 可在 FO 和 SO 状态下使用。图 3-5 是类型 2 的分组格式：

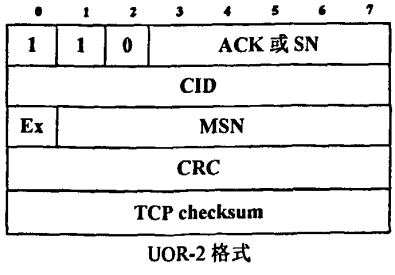


图 3-5 类型 2 的压缩分组格式

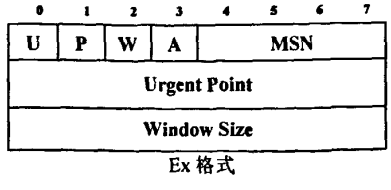


图 3-6 Ex 分组格式

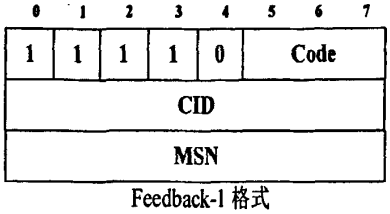
在类型 2 和类型 1 的分组中，如果 Ex＝1 则表示存在额外的字段，传输 TCP 窗口大小和紧急指针。需要附加的 Ex 分组格式如图 3-6 所示。

3.4.4 反馈分组格式

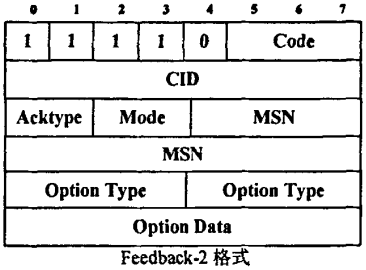
依照 ROHC 简档 1，采用所有的三种反馈格式。反馈 1 分组仅用于传输成功解压信息，编码字段（code）和 CID 字段在 ROHC 中有同样的含义，W-LSB 编码的 MSN 字段值，对应于 ACK 值。

反馈 2 分组的使用更灵活，其中，Acktype 字段有三个取值，Acktype=0 表示是 ACK 分组，Acktype=1 表示是 NACK 分组，Acktype=2 表示是 STATIC-NACK 分组。Option Type 占 4 个比特，Type=1 代表 Option Data 中是 CRC 值，Type=3 表示是 SN 值，Type=4 代表 Option Data 中是 ACK 的值。

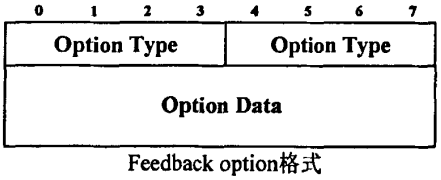
反馈 1 和反馈 2 的分组格式见图 3-7：



Feedback-1 格式



Feedback-2 格式



Feedback option格式

Type	Length	Data
1	1	CRC
2		
3	1	SN value
4	1	ACK value

图 3-7 反馈分组格式

3.5 本章小节

介绍了 TCP 数据流的特性，分析了 TCP 信头字段的变化方式，在此基础上引入主序列号 MSN 作为关键字段，编写了 TCP/IP 分组的各种压缩简档和反馈分组，并在原型系统中得到实现，所设计的方案可以通用于整个 ROHC 架构。

第四章 滑动窗口宽度的动态调整

4.1 W-LSB 算法的鲁棒性

W-LSB 算法不需要精确的同步，在 W-LSB 中对每个流，压缩器保存最近 SWW 个参考值的滑动窗口，解压器只维护最近成功解压的参考值，因此实际上 LSB 是 W-LSB 中 SWW=1 的特殊情况。对每个分组，压缩器保证压缩分组可以使用滑动窗口中的任何参考值进行解压。因此，解压文景可以使用解压缩器滑动窗口中的任何一个参考值。W-LSB 算法的鲁棒性在于最坏情况下可以容忍（SWW-1）个连续的分组丢失，而不会导致损害扩散[27]。

W-LSB 编码的鲁棒性在于如果连续分组丢失个数不超过 SWW 窗口长度，即丢失分组的个数小于参考值的个数，解压器仍然可以正确解压。因此，SWW 值在 W-LSB 编码中作用很大，对 ROHC 性能有直接影响。如果窗口宽度 SWW 过小，用做压缩的参考值就少，需要压缩的比特位数 k 短，可以提高压缩效率。但是一旦在无线链路上丢失 SWW 个以上的分组，超出了参考值可以译码的范围，解压方就不能够正确解压，影响了 ROHC 的鲁棒性。相反，如果 SWW 取值过大，虽然能够保证 ROHC 的鲁棒性，但是却降低了压缩性能。

因此，如果链路上分组丢失率比较小，选择较小的 SWW 值即可，当可能出现较大分组丢失率时，可以选择较大的 SWW。在无线环境中，分组丢失率通常由误码率 BER 引起并且可能发生连续分组丢失，如果进一步考虑移动性，信道质量可能快速改变，很难找到适合各种情况的固定 SWW 值。因此，应该动态调整 SWW 值来满足无线信道需求。

到目前为止，研究人员已经提出了很多用来估计无线信道状态的方法。文献[28]提出了一种方法，就是通过检测无线链路的状态来决定 W-LSB 滑动窗口的宽度。方法是：在解压方保持一个最近一段时间信道状态的平均抖动度量值，对于一个新接收到的分组，求出它的抖动值，如果高于链路的平均抖动值，就认为链路的状况不好，相应的滑动窗口的宽度要变大；如果低于或者等于平均值就认为链路质量比较好，滑动窗口的宽度变小。

在文献[29]中，Abhishek Das 等人采用链路状态监测器(LSM，Link State Monitor)来估计无线蓝牙链路上的信道状态。LSM 为每个信道建立并维持一个大小为 a 的窗口，保存在该信道上发送的最后 a 个分组。如果窗口中分组的差错率大于某一个阈值 A（A 是一个经过合理选择的参数），则 LSM 预测当前的信道状态为“差”，反之状态则为“好”。

在文献[30]中，Luca Marzegalli 等人提出了另外一种预测无线蓝牙链路上信道状态的方法。他们使用蓝牙协议中的逻辑链路控制和自适应协议 L2CAP 来预测信道状态及链路

上的丢包情况，为了简化算法，他们采用 L2CAP 定时超时机制而不是比特误码率来估计信道状态。L2CAP 定时超时的次数表示试图传输一个分组的逻辑连接数，如果当前 L2CAP 定时超时的次数大于上一次的定时超时次数，即传输当前分组的逻辑连接数大于上一个分组传输的逻辑连接数时，表示当前的信道状态为“差”，反之信道状态则为“好”，如果相等，则维持上一个预测结果。

Ricardo J 等人在文献[31]中提出了一种在无线 ATM 网络中预测无线链路信道状态的方法。这种估计方法以一段时间周期为间隔，在每次间隔快结束时计算收到的没有差错的帧数和有错帧数的比值。当该比值比计算的最小差错阈值还小时，表示当前的信道状态为“好”，而当该值比计算的最大差错阈值还大时，则表明信道状态为“差”。在估计过程中使用这两个阈值可以避免信道估计的结果来回地在“好”和“差”状态之间交替。

在文献[32]中，Songwu Lu 等人提出采用一步预测法(One-Step Prediction)来预测基于分组的无线网络中无线信道的状态。一步预测法通过监测前一个时间槽上的信道条件来预测当前的信道状态。在一步预测法中，不用假设无线信道的特性符合 Markov 模型，而只需假设无线信道上的差错符合突发特性且连续时间槽上的差错特性高度相关。由于基站在每个时间槽上或者传输数据分组，或者传输确认分组，因此小区中的终端可以监测每个时间槽上基站发送的分组。如果终端监测到信道被激活但是没有收到基站发送的无差错分组，则可以判断当前时间槽上信道的状态为“差”。由于相邻时间槽上信道状态高度相关，因此终端预测在下一个时间槽上信道处于和当前时间槽上的相同状态。

随着电子技术、传感技术和无线通信技术的快速发展，无线多媒体传感器网络（WMSN，Wireless Multimedia Sensor Network）引起了广泛关注。WMSN 是一种特殊的无线传感器网络，传输大数据量的音频、视频和图像等数据，需要高效地利用宝贵的无线信道资源。文献[33]中针对无线多媒体传感器网络提出了一种自适应的信头压缩机制（AAHC，An Adaptive Header Compression），基于对信道状态的仿真，动态调整压缩算法的参数，以此来动态调整信头压缩率，从而取得信头压缩与抗差错鲁棒性的平衡。

利用以上各种方法动态地判定无线链路状态，调整滑动窗口宽度，可以提高 ROHC 协议压缩的鲁棒性和压缩率，但需要增加特别的检测模块，实现起来比较复杂，增加了系统的开销，应用范围比较小。在这里，需要一种可以简单判断链路状态并且有效调整滑动窗口宽度的方法，实现起来比较容易，不额外增加系统开销。

4.2 ROHC 的反馈机制

为了保证 ROHC 解压缩的正确性，压缩文景和解压文景必须保持同步，ROHC 协议

中的反馈机制能够保证解压器及时的将解压情况通知压缩器。反馈机制有助于保证两端文景的同步，有效地避免了不恰当的压缩和错误的解压。需要注意的是，反馈机制不能用于无反馈信道的U模式，可用于O模式或R模式。

利用反馈机制，ROHC可以从解压器向压缩器发送反馈信息，主要包括正反馈（ACK分组）、负反馈（NACK分组）和静态负反馈（STATIC-NACK分组）三种类型的反馈。解压器如果成功地解压了一个压缩分组，就会发送一个正反馈分组给压缩器，通知压缩器此时两端的文景基本保持同步。当连续几个分组出现解压错误时，如果是由文景的动态部分被破坏所引起的，则解压器会向压缩器发送负反馈；若是由文景的静态部分被破坏所引起的，则发送静态负反馈给压缩器，此后压缩器会发送含有文景更新信息的分组给解压器，对被破坏的文景部分进行修复，使两端的文景重新保持同步。

对负反馈分组的处理：解压器向压缩方发送负反馈分组，请求压缩方发送具有文景更新特性（携带7-8位比特CRC校验码）的ROHC分组，以便解压方用来更新解压文景，保持压缩方和解压方的数据流同步，同时解压器根据解压状态机向低级状态转移。当压缩方接收到负反馈分组后，压缩器根据压缩状态机也要向低级状态转移，同时向解压方发送具有更新文景特性的ROHC分组。

对正反馈分组的处理：当解压器正确地解压一个带有文景更新特性的ROHC分组后，解压器就会向压缩方发送正反馈分组，表明解压器已经获得了足够的文景信息，压缩器可以转移到更高级状态，同时解压器要根据解压状态机向高级状态转移。压缩方接收到正反馈分组时，压缩器根据压缩状态机向高级状态转移，同时发送更高压缩率的ROHC分组。

4.3 动态调整的改进方法

4.3.1 O和R模式下的改进方法

在ROHC的O模式和R模式中，当解压失败时，解压器向压缩器发送NACK和STATIC_NACK负反馈信息，请求压缩器发送完整分组来更新文景，因此可以利用NACK和STATIC_NACK信息来动态调整SWW的值。

改进方法如下：

●压缩器保存压缩参考值的滑动窗口宽度设为SWW_C，解压器保存解压参考值的滑动窗口宽度设为SWW_D，为滑动窗口的宽度设定初始值。

●压缩器端接收到NACK和STATIC_NACK信息意味着解压器解压失败，表明存在分组丢失，因此当前的SWW_C值太小了，需要成倍增大SWW_C值来避免进一步的解压失

败；解压器端在发送 NACK 和 STATIC_NACK 负反馈前按照同样的规律增大。

●当连续成功压缩（或解压）的分组个数大于当前滑动窗口宽度时，意味着当前滑动窗口的宽度可以提供较强的鲁棒性，链路质量较好，在这种情况下，可以等比缩减 SWW_C（或 SWW_D）的值来获得较少的编码比特和较快的压缩速度。

4.3.2 伪代码

伪代码如下，函数中，SWW 最大值为 126，最小值设置为 14，来防止解压失败。

```
MIN_SWW=14;
MAX_SWW=126;
succeed_compress_num=0;
.....
.....
pkt_cid=receive_packet();
.....
if (packet is uncompressed)
   {compress_packet(pkt_cid);
    send_packet(pkt_cid);
    succeed_compress_num++;
    .....
   }
else if (packet is feedback) && (packet_type==NACK or STATIC_NACk)
        {sww=*2;
        if sww>MAX_SWW   sww=MAX_SWW;
        succeed_compress_num=0;
        }
.....
if succeed_compress_num>sww
   {sww=sww/2;
   if sww<MIN_SWW   sww=MIN_SWW;
   succeed_compress_num=0;
   }
```

4.4 系统仿真及结果

4.4.1 仿真工具简介

仿真中使用 NS2.3.2 软件对 ROHC 系统进行评估分析。NS2，即 Network Simulation Version 2，是面向对象的、离散事件驱动的网络环境模拟器，主要用于解决网络研究方面的问题，NS2 提供了在无线和有线网络上的 TCP、路由、多播等多种协议的模拟[34]。NS2

使用两种编程语言：OTcl（具有面向对象特性的 Tcl 脚本程序设计语言）和 C++语言，OTcl 用来建立模拟环境，配置网络具体参数信息；而 C++语言用来实现具体协议。

4.4.2 仿真系统框架

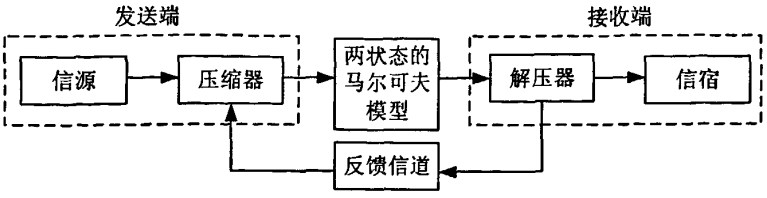


图 4-1 TCP 仿真系统框架

如图 4-1 所示，ROHC 仿真系统框架包括：压缩器、解压器、信源、信宿、马尔可夫模型信道和反馈信道。压缩器模块针对 IPv6 数据流中 TCP 信头进行压缩，同时接收来自解压器的反馈分组，根据解压情况控制自己的模式和状态转移。压缩后的 ROHC 分组在接收端交给解压器模块，解压器模块按照简档格式调用相应的解压简档还原为原始分组，解压成功后更新解压文景。无论解压成功与否，在存在反馈信道的情况下，解压器方都会向压缩器方发送反馈分组，以表明当前解压情况，目的是保证两端文景一致。反馈信道是解压器发送反馈分组的通道，和接收分组使用相同的信道。信源模块产生 TCP 业务的仿真分组并进行发送，信宿模块负责解压缩分组的接收和显示分组接收情况。马尔可夫模型信道为两状态模型：无错态和出错态。在无错态下，信道无差错，数据帧可以成功传输；而在出错态下，传送的数据帧出现错误，无法正确传输[35]。

4.4.3 仿真信源模型

仿真中信源既可以采用模拟应用程序，也可以采用流量产生器。如果采用模拟应用程序，使用 set ftp1 [new Application/FTP]和$ftp1 attach-agent $src 等命令即可设置为 FTP 信源。如果采用流量产生器，需进行相应的配置。FTP 业务模型采用 ON/OFF 模型：ON 代表数据传输过程，持续时间服从 Pareto 分布，表示 FTP 业务的数据下载时间；OFF 代表数据传输之间的间隔时间，持续时间服从 WeiBull 分布，表示上一次数据传输结束到下一次数据传输开始这段间隔时间，由 ON、OFF 相互间隔就构成了 FTP 的业务模型。使用 set pareto1 [new Application/Traffic/Pareto]的 TCL 命令来设置流量产生器，ON 使用 burst_time_参数配置为 500ms，OFF 使用 idle_time_参数配置为 200ms[28]。TCL 语言中设置命令如下：

```
set pareto1 [new Application/Traffic/Pareto]
```

```
$pareto1 set packetSize_ 1000
$pareto1 set burst_time_ 500ms
$pareto1 set idle_time_ 200ms
$pareto1 set rate_ 200K
$pareto1 set shape_ 1.5
```

4.4.4 无线信道模型

无线信道的特点：

（1）高误码率：无线链路易出错，其比特误码率变化范围为 10^{-6}~10^{-3}，最高甚至可达 10^{-2}；

（2）长往返时延：无线链路的往返时延时间可能大到 100~200ms；

（3）衰落特性：无线信道中存在多种衰落，包括多径衰落和慢衰落等；

（4）差错突发性：无线链路上的差错呈现出明显的突发特性。

另外，在无线移动信道下，由于衰落等因素的影响，信道是有一定记忆性的。为此，我们采用能够模拟无线信道差错特征且能反映无线信道记忆特性的马尔可夫（Markov）模型作为信道模型，双态 Markov 模型如图 4-2 所示：

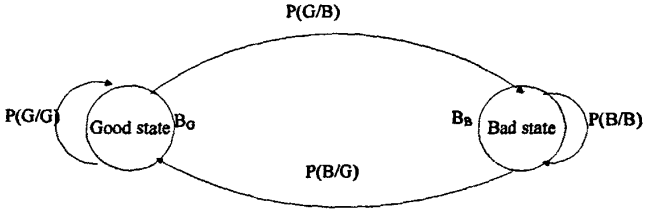


图 4-2 MarKov 模型

双态 Markov 模型将信道分为两种状态：出错态（Bad state）和无错态（Good state），出错态下表示传送的分组发生差错，无错状态下分组成功发送，信道以不同的概率在这两种状态间转移。这两种状态的信道误码率（BER）和持续时间假定均符合均匀分布，均值分别为 B_b 和 B_g，T_b 和 T_g，这四个参数的值可以调整，以对应不同的信道状态。这里，我们主要是仿真无线信道的高误码率和长往返时延特性。可以定义 B_b 和 B_g 的数值在 0~1 之间，例如 B_b=0.0091，B_g=0.04，T_b=0.0075，T_g=0.00375[28]。TCL 语言中设置命令如下：

```
set tmp1 [new ErrorModel/Uniform 0.0091 pkt]
set tmp2 [new ErrorModel/Uniform 0.04 pkt]
set m_states [list $tmp1 $tmp2]
```

```
set m_periods [list 0.0075 0.00375]
set m_trunit pkt
set m_sttype time
set m_nstates 2
set m_nstart [lindex $m_states 0]
set em [new ErrorModel/MultiState $m_states $m_periods $m_transmx]
$m_trunit $m_sttype $m_nstates $m_nstart]
```

无线节点的配置命令如下：

```
set val(chan) Channel/WirelessChannel
set val(prop) Propagation/TwoRayGround
set val(netif) Phy/WirelessPhy
set val(mac) Mac/802_11
set val(ant) Antenna/OmniAntenna
```

4.4.5 仿真模式限定

TCP 数据流是双向流，不仅在连接建立阶段需要三次往返握手，而且在以后的数据发送中，还需要返回确认的分组，因此 TCP 流一般使用双向链路。因此，我们将仿真限定在 O/R 双向运行模式，而不考虑 U 模式的运行情况[36]。

O 模式下（反馈是可选项），使用 IR-CR 分组进行压缩和解压初始化，在 IR-CR 状态成功发送 N_2 个分组以后，压缩器转移到 CO 最高压缩状态。对于解压器只要成功解压一个分组，就从 NC 转移到 FC 状态。向下跃迁使用反馈原则，在 FC 状态，如果连续 K_1 个分组解压失败，解压器转移到 SC 状态且压缩器转移到 IR-CR 状态；在 SC 状态连续 K_2 个分组解压失败后，解压器转移到 NC 状态且压缩器转移到 IR 状态。

对于 R 模式，主要使用反馈原则。压缩器初始工作在 IR 状态，只要收到一个 ACK 分组就向上转移到 IR-CR 状态，在 IR-CR 状态收到一个 ACK 分组就向上转移到 CO 状态。解压器仍然是成功解压一个分组，就从 NC 转移到 FC 状态。向下跃迁使用负反馈，在 FC 状态连续 K_1 个分组解压失败，解压器转移到 SC 状态且发送负反馈使压缩器转移到 IR-CR 状态；如果在 SC 状态连续 K_2 个分组解压失败，解压器转移到 NC 状态且发送负反馈使压缩器转移到 IR 状态。

4.4.6 ROHC 仿真实现

ROHC 协议在 NS2 中的实现框架如图 4-3 所示：

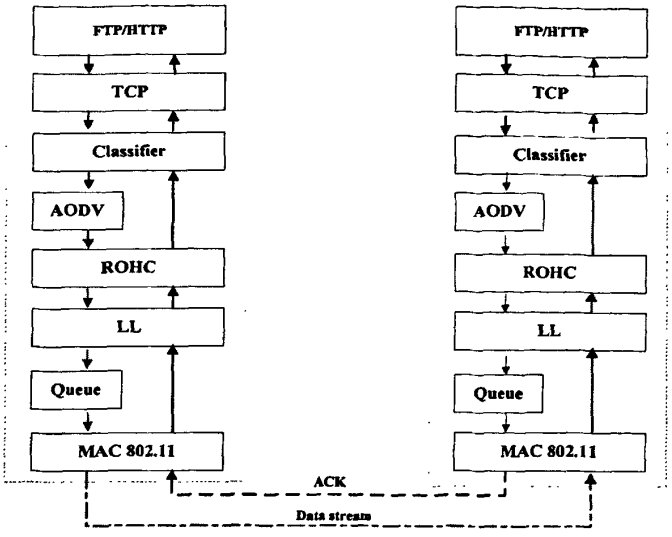


图 4-3 ROHC 仿真实现框架

（1）在~ns2.3.2/common/packet.h 文件中增加 ROHC 分组类型的定义，对其分组类型和分组名进行绑定。首先，在 enum packet_t 中添加新增的 ROHC 分组类型值，定义为 PT_ROHC；之后，在 class p_info 类的 p_info()构造函数中增加一行，给出新增分组类型的名字：Name_[PT_ROHC]= “ROHC”。

（2）打开~ns2.3.2/tcl/lib/ns-packet.tcl 文件，添加新定义的分组名字 ROHC，名字与前面定义的 PacketHeader/ROHC 一致，添加方式如下：

```
Foreach prot{
        ……
        AODV
        ROHC
        ……
}{add-packet-header $prot}
```

修改~ns2.3.2/tcl/lib/ns-default.tcl 文件，对 ROHC 的初始值进行设置。如果不进行设置，系统在与 OTcl 绑定时会出现错误。

（3）编写 rohc.cc 和 rohc.h，这是 ROHC 的实现部分。在 rohc.h 中定义 rohc 信头格式，在 rohc.cc 中将 ROHC 类和 TCL 部分中的 Agent/ROHC 绑定在一起，代码如下：

```
static class RohcClass:public TclClass
```

```
{
public:
RohcClass():TclClass("Agent/ROHC"){}
TclObject *create(int argc, const char*const* argv){return (new
ROHC());}
}class_rohc;
```

（4）修改~ns2.3.2/Makefile 文件，增加 rohc.o 目标文件。

（5）重新编译 NS。

（6）编写 rohc 仿真需要的 tcl 程序，运行后编写 gawk 脚本抽取数据进行统计。

4.4.7 评估压缩性能的方法

信头压缩有很多性能参数，它们都用于不同的评定环境，侧重于评估某些特定的压缩性能。常见的性能参数有压缩效率（CE，Compression Efficiency）、鲁棒性和压缩透明性(CT，Compression Transparency)。

压缩效率取决于压缩方案中有多少信头大小得到了缩减，可以使用平均压缩信头长度（ACL，Average Compressed header Length）来评价，单位是字节。ACL=成功压缩后（或成功解压前）的信头总长度 / 成功压缩（或解压）的分组个数。ROHC 一般可以将信头压缩到 3~5 个字节，在最高状态下信头可以压缩到 1 个字节。压缩效率也可以使用压缩率来评价，压缩率＝信头压缩掉的字节总数 / 总的发送字节数。

ROHC 协议对 IP 分组信头进行压缩，除具有很高的压缩率外，还应该具有很好的鲁棒性，使得 ROHC 协议能够适应高差错率、低带宽和长时延的无线链路环境。ROHC 协议的鲁棒性一般采用分组丢失率（RLR，Rohc Loss Rate）来评价，RLR＝丢失的分组个数 / 总的发送的分组个数。强的鲁棒性可以容忍链路上丢失分组和残留误码错误，即使进行信头压缩，在解压信头时不会丢失附加分组或引入附加错误。

压缩透明性是衡量解压缩后所能保证的分组信头内容和原来分组信头内容一致性的程度。

其它性能评估参数还有系统开销（Overhead），指经过压缩，信头长度占整个分组长度的百分比值。计算公式＝压缩后的信头大小 / 压缩后的分组大小，压缩后的分组大小包含压缩后的信头大小和净荷大小。

4.4.8 仿真结果分析

动态调整滑动窗口宽度 SWW 的目的在于增加系统的自适应鲁棒性，由于引入了一些必要的控制单元，要兼顾系统资源的消耗，仿真中采用的评价标准参考了鲁棒性和系统开销。当滑动窗口宽度较小时，由于滑动窗口内保存的参考值较少，压缩时 W-LSB 编码的最低有效位比特也较少，可以获得较快的压缩速度，系统的开销较小，但系统的鲁棒性不强，一旦分组丢失的个数超过滑动窗口宽度，就无法完成正确的解压。当滑动窗口宽度较大时，虽然可以提高系统的鲁棒性，但由于滑动窗口内保存了过多的参考值，在进行 W-LSB 编码时会引起很大的系统开销。

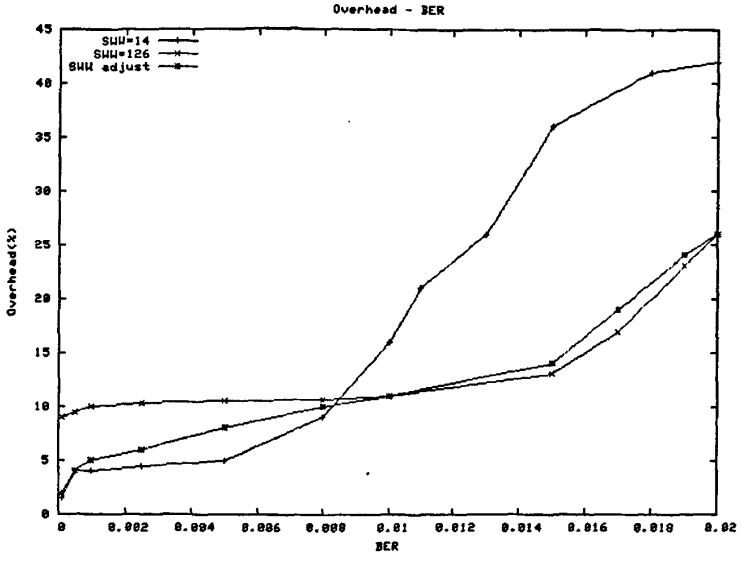


图 4-4 不同 BER 下系统的开销

图 4-4 是不同误码率 BER 下系统开销的仿真结果，从以上结果可以看出：

（1）在 BER<0.008 时，动态调整 SWW 方法和 SWW＝14 的固定设置可以获得较低的系统开销，低于 SWW＝126 的系统开销。这说明链路状态良好，误码率较低的情况下，动态调整 SWW 设置的系统可以很快进入高压缩率状态，减少系统开销。

（2）当 BER>0.008 时，随着误码率的提高，系统开销开始增大，采用 SWW=14 设置的系统开销开始急剧增大，这是因为随着误码率 BER 的提高，需要更多的更新或同步文景，发送未压缩的 IR 分组或低压缩的 IR-DYN 分组较多，在这种情况下鲁棒性较差的 SWW＝14 方法需要更多的系统开销。鲁棒性较强的 SWW＝126 和动态调整 SWW 的方法，由于抗差错率强，引起的系统开销增长不大。

以上分析表明，动态调整 SWW 的方法在低误码率情况下接近于 SWW=14 的较低系

统开销，在高误码情况下系统开销接近强鲁棒性的 SWW=126 方法，在误码率波动较大的整个区间上，动态调整 SWW 的方法具有平均较低的系统开销，可以较好地适应发生突发变化的无线链路。

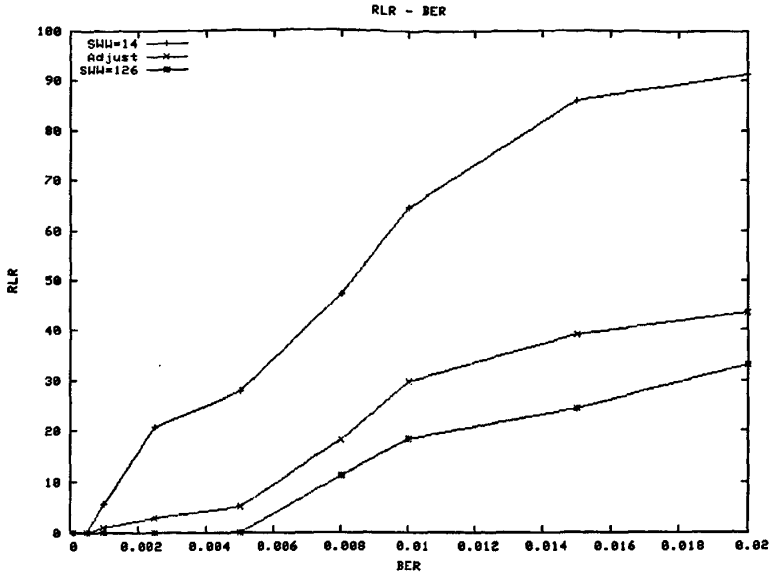


图 4-5 不同 BER 下系统的分组丢失率

图 4-5 是不同误码率 BER 下 ROHC 分组丢失率 RLR 的仿真结果，从图中可以看出：随着误码率 BER 的增加，ROHC 的分组丢失率 RLR 随之上升(鲁棒性下降)，固定 SWW=14 的方法在 BER>0.002 时，鲁棒性下降较快。随着误码率的变化，动态调整 SWW 的方法和固定 SWW＝126 的方法基本保持了较低的分组丢失率，有较强的抗差错鲁棒性。

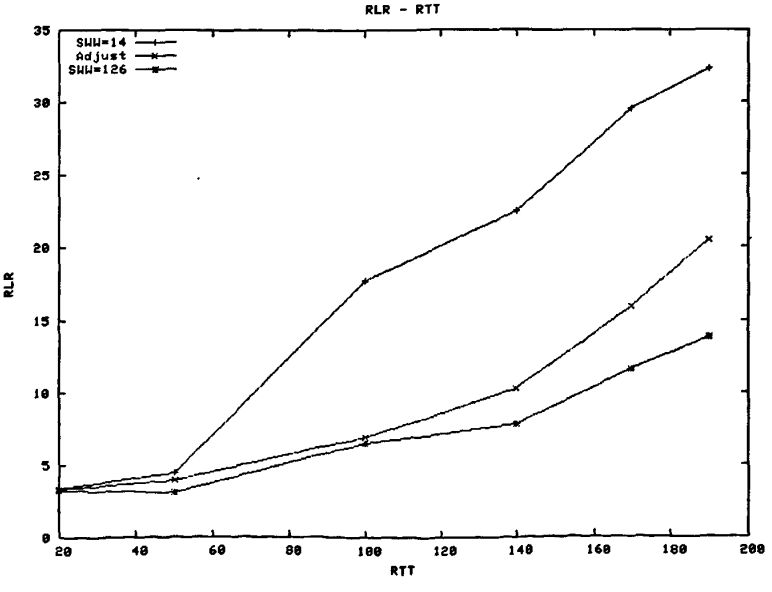


图 4-6 不同 RTT 下系统的分组丢失率

图 4-6 是在不同往返时延 RTT（RTT 取 20、50、100、140、170、190ms）下，分组丢失率 RLR 的仿真结果。可知随 RTT 的不断增加，ROHC 的分组丢失率也急剧增大，动态调整 SWW 方法的鲁棒性介于 SWW=14 和 SWW=126 的性能之间。

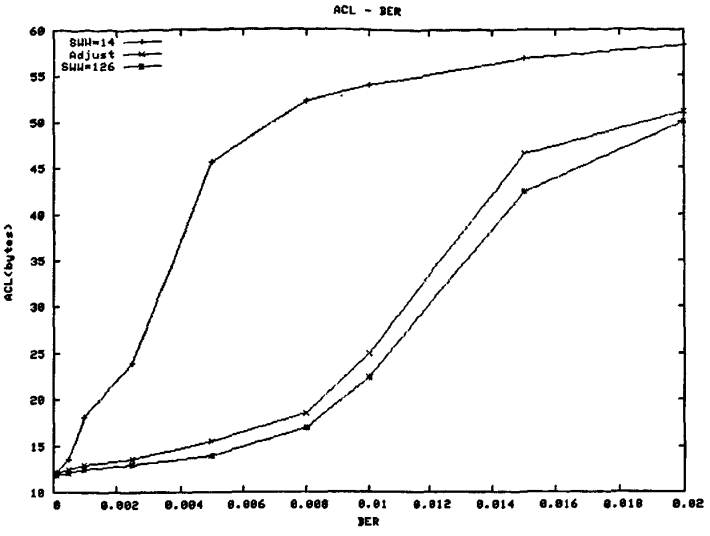


图 4-7 不同 BER 下系统的平均压缩信头长度

图 4-7 是不同误码率 BER 下 ROHC 分组的平均压缩信头长度，从图中可以看出：误码率 BER 增大时，平均压缩信头长度也随着变大（压缩效率下降），SWW=14 的方法由于鲁棒性较差，压缩效率下降很快，而动态调整 SWW 的方法和 SWW＝126 的方法压缩效率下降较为缓慢，当 BER>0.01 后，压缩效率下降较快，这是因为当解压失败时，压缩器需要发送更多的未压缩分组或静态分组来更新文景。

强鲁棒性的 SWW=126 方法适合于高误码率链路，而在低误码率链路上 SWW＝14 的方法引起的系统开销较小，而动态调整 SWW 的方法较好地综合了以上两种方法的优点，在高误码率链路上具有较强的鲁棒性，在低误码率链路上具有较低的系统开销，可以随链路状态进行适应性调整，有更宽泛的适应范围。

4.5 本章小结

ROHC 系统中建议或经过仿真优化后的滑动窗口宽度 SWW 都是固定设置，只适合特定的使用环境。如果设定的 SWW 值较小，可以提高系统的压缩速度，但不适应误码率较高的无线链路；而设定的 SWW 值过大时，系统的鲁棒性增强，提高了抗差错率，但引入了较大的系统开销。动态调整 SWW 值的方法兼顾了 SWW 值较小时的低系统开销和 SWW 值较大时的强鲁棒性，可以很好地适应误码率突发和长时延的无线链路。

同时，动态调整 SWW 值的方法没有改变压缩和解压状态变迁逻辑，也没有引入附加

的控制信息和开销，与复杂的信道检测无关，具有独立性，应用范围较广。

第五章 ROHC的测试信源和接口

5.1 ROHC测试信源的需求

目前关于信头压缩性能的研究典型地都假设某些特殊情况，比较理想。而实验条件下生成的信头内容可能和实际环境中不同，因为大多数的研究都无法保证它们能产生实际环境中的信头，所以在信源上都假定了理想的操作环境（例如处理非并发信息流）。因为这些特点，实验环境下生成的分组信头会有很高的压缩效率。

为了处理信源过于理想的问题，文献[38]开发了一个信源模型来研究信头压缩，通过变更模型框架中信源和信道模型的参数，能模拟不同的信源和配置方案。框架用5个随机进程作为其重要构件：信源进程、信源-压缩器信道、压缩器、压缩器-解压器信道、解压器。框架中通过引入信源进程和信源-压缩器信道，获得了影响性能主要构件的完整说明。文献[38]通过使用一种新的方式来研究性能参数之间的平衡相关性，从而对信头压缩性能参数的定义和理解提供了新的视角。

在ROHC简档的编写和调试中，在没有完成全部简档的编写前，经常需要随时测试当前简档的编写或修改是否有效，这时很需要有临时的测试信源，随时检验简档的压缩和解压缩功能，进行下一步的修改。另外一方面，对于TCP流，在数据分组传输前需要建立连接，而接收端接收到TCP分组后要发送确认分组（ACK），在TCP简档的修改过程中，由于简档的不完善（缺少某些压缩简档或简档中包含错误），这时ROHC压缩器处于调试阶段，无法保证握手分组或确认分组的正确到达，从而可能产生死机或TCP重传，导致无法提取系统的压缩信息进行分析和进一步完善，影响了简档的编写和测试。特别是，某些特殊数据流（如ESP分组）的测试，信源的产生比较困难，也阻碍了简档的测试。

在这里，需要一个可以随时产生的测试信源，不需要建立连接就可以模拟发送需要的各种分组格式，信源可以随时终止，不会产生不良的终止后果。

5.2 产生测试信源的方法

ICMP（Internet Control Message Protocal）同IP协议一样工作在ISO模型的网络层，是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息，它的主要作用是主机探测、路由维护选择和流量控制。用于测试网络连接的Ping命令，是系统自带的一个可执行命令，发送的就是ICMP分组格式。

ROHC对ICMP分组是不进行处理的。ROHC压缩器端对所有的发送分组进行检测，

判断分组类型，如果是 ICMP 分组就调用原来的 hard_start_xmit()函数直接发送；如果是需要处理的分组，按照对应的简档格式进行压缩，如图 5-1 所示：

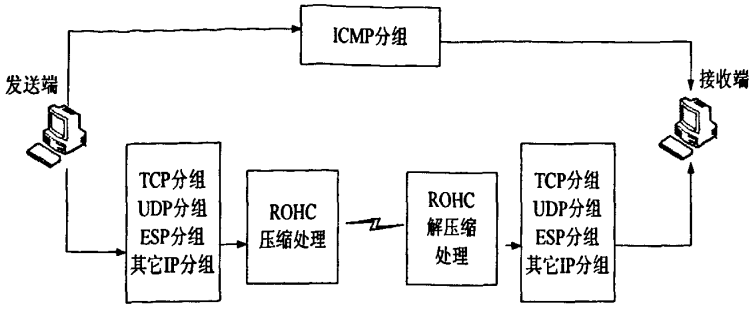


图 5-1 ROHC 对分组的分类处理

在终端窗口中使用 Ping 命令就可以随时产生 ICMP 分组，方便地更改接收端、发送次数和负载大小，非常灵活，类似我们对测试信源的要求。如果改变了 ICMP 分组的分组类型标识（ip→nexthdr），系统就要按照改变后的分组类型调用相应的简档，进入对应的压缩处理流程，经压缩后发送到接收端。

为了方便地产生测试信源，我们在 ROHC 压缩器端进行了修改，修改方法如下：

●在压缩器 rohc_compressor.c 的 get_profile（）函数中，根据 ip->nexthdr 数值判断分组类型。如果为 ICMP 类型，将简档标识（profile_id）改成需要测试的其它简档标识；

●按照需要的分组格式对 skb_buff 进行改写，产生指定格式的测试分组；

●get_profile（）函数会返回更改后的简档标识，ROHC 压缩器按照对应的简档进行压缩处理，然后发送出去；

●在 ROHC 压缩器中还需要设置一些整型变量，按照一定的运算规则进行计算，用于产生分组中不断变化的字段（如 TCP 分组中的 SN 字段或 RTP 分组中的 Timestamp 字段）；

●不需要处理负载数据，负载的大小由 Ping 命令中参数指定。

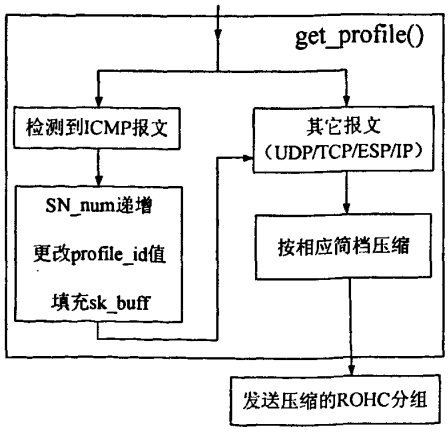


图 5-2 修改后的 get_profile 函数流程

修改后的 get_profile（）函数运行流程如图 5-2 所示。

5.3 测试信源的使用

ping 命令方便实用，可以随时发起压缩/解压缩请求，测试快捷方便，同时 ping 命令中含有的丰富参数可以产生不同类型的测试分组。

使用 ping 命令即可产生 ICMP 分组，其中 ping 命令中的 IP 地址指明了 IP 分组中的目的地址。-n 参数指定了连续发送数据分组的个数，使用数值 1 可以简单快速地测试压缩/解压缩功能是否正常。-s 参数指定了要发送的数据字节数，可以随意地调节 s 参数，以获得所需的负载大小。发送端会将 ICMP 分组更改为其它分组格式，按照设计好的简档进行压缩，然后发送到接收端。

接收端在接收到 ROHC 压缩分组后，按正常程序进行解压，然后向上层递交，由于不存在实际的接收端，分组自然丢失，不需要发送确认分组，即使链路上出现分组丢失也不影响后续测试的持续进行。即使突然终止，也不会产生不良结果。

以上方法在 Linux2.6.19 操作系统下，通过修改 ROHC 协议的原型系统得到了实现，在真实环境中进行了验证。

5.4 测试信源的优点

通过修改 ROHC 压缩器的部分函数，配合系统的 Ping 命令，产生了所需要的 ROHC 测试信源，得到了真实环境的验证。这种方法有以下优点：

（1）测试信源的使用方法灵活，可以控制测试分组发送的个数和负载大小，这些只需要在 Ping 命令中调整一下参数即可得到；

（2）由于这种方法直接修改了低层的 skb_buff 数据值，只要知道分组的数据格式，就可以产生任何类型的测试信源，而不需要考虑这种协议在协议栈中的位置；

（3）数据是单向发送的，不需要建立连接（针对 TCP 简档），避免了死机等现象的发生。对于 TCP 测试信源，不需要建立连接，也不要确认分组的反馈，测试分组可以持续发送；

（4）任何情况下终止测试，都不会引起不良的结果，从而影响系统的稳定。

5.5 用户接口

●ROHC 管理接口

定义了 ROHC 运行时的关键参数：ROHC 分组类型（ETH_P_ROHC）、模式跃迁控制（CHANGE_MODE）、窗口宽度（ROHC_WINDOW_WIDTH）、文景标识符的大小（MAX_CID）、各种分组的简档类型和压缩/解压类型、反馈类型（ACK、NACK、STATIC_NACK）等。

●ROHC 调试接口

提供了内核信息输出接口，供调试使用，包括按规范格式打印 skb_buff 内容（skb_buff 内容分为四部分：压缩前 Sending、压缩后 Compressed、接收端 Received、解压后 Decompressed），输出系统中分组压缩/解压的处理过程信息、显示系统的状态和模式变迁、统计压缩/解压分组个数、计算压缩率和平均信头长度（ACL）等。

●系统运行参数的输出

采用 proc 文件系统的方法，在系统的/proc 文件系统下面建立一个输出，主要输出的统计信息是当前所有以太网接口的压缩参数（当前活跃的压缩器和解压器的所有状态信息）。

5.6 统计接口

每个压缩器和解压器都含有 struct rohc_dev 结构类型，struct rohc_dev 类型含有以下字段，用于统计各种需要的信息：

```
unsigned long   total_cm_number;                    //压缩 ROHC 分组的个数
unsigned long   total_rc_feedback_number;           //接收到反馈分组的个数
unsigned long   total_dc_number;                    //解压 ROHC 分组的个数
unsigned long   total_sd_feedback_number;           //发送确认分组的个数
unsigned long   total_cm_bytes, total_dc_bytes;     //压缩和解压的字节数
unsigned long   total_rc_feedback_bytes, total_sd_feedback_bytes;
                                                    //接收到反馈分组的字节数和发送反
                                                    馈分组的字节数
unsigned long   total_receive_bytes, total_send_bytes; //接收到字节数和发送字节数
unsigned long   total_rohc_loss_number, total_ipv6_loss_number;
                                                    //在解压端丢失的 ROHC 分组个数和
                                                    丢失的字节数
```

压缩器压缩、解压器解压和发送反馈分组时对以上字段进行累计更新，可以统计总共发送的 ROHC 分组个数和字节数、解压的分组个数和字节数、反馈分组和字节数以及丢失

的 ROHC 分组个数和字节数。

5.7 本章小节

在编写 TCP/IP 分组的压缩简档时，需要随时测试压缩简档的正确与否，通过在压缩器端增加处理函数，满足了 ROHC 系统测试信源的需求，测试信源可以控制测试分组发送的个数和负载大小，可以产生任何类型的测试信源，尤其是在 TCP 压缩简档的测试中，不需要建立连接，也不需要确认分组的反馈，测试分组可以持续发送。在任何情况下终止测试，都不会引起不良的结果，保证了系统的稳定。

第六章 结束语

6.1 本文工作总结

本文研究无线 IPv6 互连网中 TCP 信头压缩技术及性能，主要包括三个方面的研究内容：编写了 TCP/IP 分组的压缩简档；提出了动态调整滑动窗口宽度的方法；设计了 ROHC 的测试信源。

本文取得的研究成果如下：

（1）编写了 TCP/IP 分组的压缩简档

TCP/IP 分组需要使用关键字段来获得最大的压缩比，如果关键字段已知，其余字段变量全部可以推断。对于 TCP 流的情况，引入主序列号 MSN 字段，参照 ROHC 分类标准，给出了 TCP 信头字段的分类方式，采用 W-LSB 编码方法，编写了 TCP/IP 分组的压缩简档，遵循 ROHC 简档规范，所设计的方案可以通用于整个 ROHC 架构。

（2）提出了滑动窗口宽度的动态调整方法

滑动窗口宽度 SWW 指保存压缩/解压缩的参考值个数，SWW 值较小时，可以提高压缩效率，但无线链路上丢失 SWW 个以上的分组时，会超出了参考值译码的范围，降低了 ROHC 的鲁棒性；相反，如果 SWW 取值过大，虽然能够保证 ROHC 的鲁棒性，但是 W-LSB 编码中的最小比特位数 k 会变大，降低了压缩率。

利用解压失败时向压缩器发送的负反馈信息，动态调整滑动窗口的宽度，比较适合容易发生突发错误的无线链路，这种方法没有改变压缩和解压状态变迁逻辑，也没有引入附加的控制信息和开销，所以，动态调整 SWW 的方法较好地兼容了强鲁棒性和低系统开销的优点。

（3）设计了 ROHC 的测试信源

在 ROHC 简档的编写中，经常需要测试简档的编写或修改是否有效，这时特别需要有灵活的测试信源，随时检验简档的压缩和解压缩功能，进行下一步的修改。在 TCP 压缩简档的修改中，由于处在调试阶段，可能无法保证握手分组或确认分组的正确到达，从而可能产生死机或 TCP 重传，影响了测试的正常进行。

所设计的测试信源可以产生适用于任何类型的测试分组，使用简便，分组格式不限。针对面向连接的 TCP 流测试，在不需要事先建立连接和回送确认分组的情况下，TCP/IP 流的 ROHC 压缩/解压缩测试可以持续进行。

6.2 下一步展望

（1）研究 Mobile IPv6 中的信头压缩算法

Mobile IPv6 是一种计算机网络通信协议，它能够保证计算机移动过程中在不改变现有网络 IP 地址、不中断正在进行的网络通信及不中断正在执行的网络应用的情况下，实现对网络的不间断访问。Mobile IPv6 协议中使用的复杂协议信头引入了较大的额外开销，信头本身可能达到 100 个字节，对其进行压缩可以节省传输带宽。

（2）研究乱序接收情况下的 TCP 性能

IETF 工作组于 2008 年 4 月提出了 RFC5225（ROHC 版本 2），比 RFC3095 有一个更广泛的应用范围，对压缩/解压缩端行为的一些规则和算法进行了简化，例如对压缩端点间的发送顺序没有严格要求，即使接收分组和发送分组不同，仍然能有一个被成功解压的合理概率。但对于 TCP 简档仍然没有太多涉及，此处应该可以研究补充。

（3）研究 ROHC 协议的协商机制

无线通信系统中，必须在移动通信设备底层确定一个良好的 ROHC 协议协商机制和无线链路状态评估算法，从而可以使 ROHC 协议能够根据不同的业务和不同的链路状况进行模式转移和关键参数的协商设定，从而提高 ROHC 协议的鲁棒性和压缩性能。

致 谢

本论文的研究工作顺承了中兴通信的项目，在此表示感谢。

作者在硕士研究生期间，得到了导师张载龙的精心指导和孜孜不倦的教悔。张老师治学严谨，循循善诱，对工作严格要求，教会我要脚踏实地做好科研工作，以及永不疲倦的工作精神，受之教导受益匪浅。在张老师的带领下，302 教研室一直保持一种良好的科研氛围。两年半的指导学习，使我在学习新知识之外还接受了全新的思想观念，领会了基本的思考方式。张老师还对我的生活做指导，使我明白了许多待人接物、为人处世的道理。

衷心感谢 302 教研室里的所有同窗好友，在科研上大家一起学习，在生活上互相交流，我们一起度过了紧张而又丰富的研究生生活。

感谢妻子杜玉新和儿子李嘉宁，感谢她们对我学习的一贯支持和生活上无微不至的关怀。在即将踏上工作岗位之际，祝家人身体安康，万事如意！

再次衷心感谢教导和帮助过我的各位老师、同学和朋友！感谢审查论文的各位专家和评委，并希望各位专家老师提出宝贵的意见。谢谢您们！

参考文献

[1] D. E. Taylor, A. Herkersdorf, A. Doring. Robust header compression (ROHC) in next-generation network processors[J]. IEEE ACM Transactions on Networking, 2006, 13(4): 755-768.

[2] F. H. P. Fitzek, S. Rein, P. Seeling, et al. Robust header compression (ROHC) performance for multimedia transmission over 3G/4G wireless networks[J]. Springer Wireless Personal Communications, 2005, 32: 23-41.

[3] D. J. Farber, G.. S. Delp. A thinwire protocol for connecting personal computers to the INTERNET. RFC914, September, 1994.

[4] V. Jacobson, Compressing TCP/IP Headers for Low-Speed Serial Links. RFC1144, February, 1990.

[5] M. Degermark, B. Nordgren and S. Pink, IP header compression. RFC2507, February, 1999.

[6] S. Casner, V. Jacobson, Compressiong IP/UDP/RTP Headers for Low-Speed Serial Links. RFC2508, February, 1999.

[7] A. Giovanardi, G.. Mazzini, M. Rossi, et al. Improved header compression for TCP/IP over wireless links[J]. IEEE Electronics Letters, November (2000), 1958–1960.

[8] H. Liao, Q. Zhang, W. Zhu, Y. Q. Zhang, et al. TCP aware robust header compression (TAROC). INTERNETDRAFT, 2001:96-102.

[9] R. Price, S. McCann, A. Surtees, et al. Framework for EPIC-LITE. INTERNET-DRAFT, 2002: 156-162.

[10] C. Bormann, C. Burmeister, M. Degermark, et al. RObust Header Compression (ROHC): Framework and Four Profiles: RTP, UDP, ESP, and Uncompressed. RFC3095, July, 2001.

[11] L. E. Jonsson, K. Sandlund, G. Pelletier, P. Kremer. RObust Header Compression (ROHC): Corrections and Clarifications to RFC 3095. RFC4815, February, 2007.

[12] P.Camarda, S.Petrizzelli. Performance analysis of a new header compression scheme for TCP streams in IP based wireless networks[J]. IEEE, 2002, 5(1): 276-281.

[13] G. Boggia, P. Camarda, V. G. Squeo. ROHC+: A new header compression scheme for TCP streams in 3G wireless systems[J]. proceedings of IEEE ICC, 2002, 9(2): 156-161.

[14] C. Y. Cho, S. K. Hazra, Winston K. G.. Seah. Exploiting inter-flow redundancy: context replication in ROHC-TCP[C]. IEEE Global Telecommunications Conference, 2004, 5:

1121-1127.

[15] L. E. Jonsson, G. Pelletier, K.Sandlund, Ericsson. The Robust Header Compression(ROHC) Framework. RFC4995, July, 2007.

[16] L. E. Jonsson. RObust Header Compression (ROHC): Requirements on TCP/IP Header Compression. RFC4163, August, 2005.

[17] G. Pelletier, K. Sandlund, L. E. Jonsson, M. West. RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP). RFC4996, July, 2007.

[18] G. Pelletier. RObust Header Compression (ROHC): Context Replication for ROHC rofiles. RFC4164, August, 2005.

[19] M.west, S.McCann. TCP/IP Field Behavior. RFC4413, March, 2006.

[20] G. Pelletier, K. Sandlund, Ericsson. RObust Header Compression Version2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP-Lite. RFC5225, April, 2008.

[21] Wu YC, Huang K, Zheng JP, et al. An adaptive robust TCP/IP header compression algorithm[J]. Computer Research and Development, 2005, 42(4): 655-661.

[22] Ana Minaburo, Loutfi Nuaymi, Kamal Deep Singh. Configuration and analysis of Robust Header Compression in UMTS[C]. Personal, Indoor and Mobile Radio Communications (PIMRC 2003), 2003, 6: 76-78.

[23] A. C. Minaburo, K. D. Singh, L. Toutain, L. Nuaymi. Proposed behavior for Robust Header Compression over a radio link[C]. IEEE International Conference, 2004, 9: 452-457.

[24] Samiha Ayed, Ana Minaburo, Laurent Toutain. Enhancing Robust Header Compression over IEEE 802 networks[J]. Wireless and Mobile Computing, Networking and Communications, 2006, 6(3): 836-839.

[25] H. Hannu, Z. Liu. Signaling Compression (SigComp). RFC3320, Jan, 2006.

[26] N. Giouroukos, G. Orphanos. An implementation of ROHC for TCP/IPv6[J]. International Workshop on Wireless Ad-Hoc Networks, 2004: 212-216.

[27] Kazem Sohraby, Chonggang Wang. Performance of packet header compression and context replication in wireless/mobile systems[J]. IEEE, 2006: 1-7.

[28] Wang. B., Schwefel, K.C, et al. On Implementation and Improvement of Robust Header Compression in UMTS[C]. IEEE PIMRC, 2002: 1151-1155.

[29] A. Das, A. Ghose, V. Gupta, et al. Adaptive Link-level Error Recovery Mechanisms in Bluetooth[C]. Personal Wireless Communications 2000 IEEE International Conference, 2000,

12: 85-89.

[30] L. Marzegalli, M. Masa, M. Vitiello. Adaptive RTP/UDP/IP Header Compression for VoIP over Bluetooth[J]. European Wireless, 2002, 11(3): 22-28.

[31] Ricardo J, Anchez S, Fadi F. Design and Evaluation of an Adaptive Data link Control Protocol for Wireless ATM Networks[C]. Proceedings of IEEE GLOBECOM98, Australia, 1998, 11: 763-769.

[32] S. Lu, V. Bharghavan, R. Srikant. Fair scheduling in wireless packet networks[J]. IEEE/ACM TRANSACTIONS ON NETWORKING, 1999, 7(4): 312-317.

[33] 周新运，孙利民，皇甫伟. 无线多媒体传感器网络中一种自适应的信头压缩机制[J]. 软件学报, 2007, 5(1): 1122-1129.

[34] NS2 Official site, http://www.isi.edu/nsnam/ns/.

[35] Wang HS, Moayeri N. Finite-State Markov channel: A useful model for radio communications channels[J]. IEEE Trans on Vehicular Technology, 1995, 44(1):163-171.

[36] Chia Yuan Cho, Yong Huat Chew, Winston Khoon. Modeling and analysis of Robust Header Compression performance[C]. 2005 IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks, 2005, 12: 192-197.

[37] A. Yiannakoulias, E. Kuehn. Evaluation of header compression schemes for IP-based wireless access systems[J]. IEEE Wireless Communications, 2005, 3: 68-74.

[38] Chia Yuan Cho, Winston Khoon Guan Seah, Yong Huat Chew. A framework and source model for design and evaluation of Robust Header Compression[J]. Computer Networks, 2006, 50(15): 2676–2712.

发表论文

李祥中，张载龙，“工业控制中实时嵌入式系统的设计和应用”，计算机应用研究，2008 年 12 月，Page(s): 764–767.