卒業論文

題目

輻輳制御を考慮したコネクション指向インターネットワーキングプロトコルの設計と実装

指導教授

所 真理雄 教授

慶應義塾大学

理工学部 電気工学科

1995 年度

69108008

成田 多良

論　文　要　旨

学科	電気工学	学籍番号	69108008	氏名	成田 多良

(論文題目)

輻輳制御を考慮したコネクション指向
インターネットワーキングプロトコルの設計と実装

近年, コネクションレス型のネットワーク層プロトコルは, 輻輳制御の観点から実時間性を持つ連続データの配送には不適切であることが指摘されている [Top90, ZDE*93]. そこで効率的に輻輳制御を行なうために, いくつかのコネクション指向ネットワーク層プロトコルが開発されてきた [Top90, SJC92]. それらの多くは, ネットワーク資源を予約して占有利用することにより輻輳を防いでいるが, 資源占有の結果として, ネットワーク資源の利用に無駄が生じやすい. また, 資源予約をサポートしないネットワークを経由する通信が不可能であり, インターネットにおける接続性が低下するという問題を生じる.

我々はネットワーク層の新しいコネクション指向インターネットプロトコル, "CIP" (Connection-oriented Internet Protocol) を提案する. CIP は二つの大きな特徴を持つ.

- 直接的な輻輳回避を行なうために, コネクションの経路上に存在する各ゲートウェイ同士がスライディングウィンドウ方式のホップ間フロー制御を用いて輻輳の検出, 回避を行なう.
- コネクションの経路上に存在する各ゲートウェイは従来の IP ゲートウェイを含むことができるため, CIP は IP と共存し, そして IP に関連する既存の資産を活用できる利点を持つ.

我々は CIP を Mach 3.0 上にユーザアプリケーションとして実装し, ネットワーク資源の効率的利用という観点から評価, 検討する.

さらに, 将来的にネットワーク帯域幅が増加し, ホップ間のフロー制御のオーバーヘッドが相対的に増加していくことを考慮して, 我々は代替の制御機構を準備する目的でデータの転送率制御, バッファ空間の予約を CIP のコネクション機構上に試験的に実装し, それらの有効性について検討する.

慶 應 義 塾 大 学 理 工 学 部

◎ 内容の要旨は約 600 字

Summary of Bachelor's Thesis

Course	Student No.	SURNAME, Firstname
Electrical Engineering	69108008	NARITA, Tomio

Title

Design and Implementation of a Connection-Oriented Internetworking Protocol based on Congestion Control

In recent years, it has come to light that connection less internetworking protocols are inadequate for real-time continuous data delivery in terms of congestion control[Top90, ZDE*93]. For efficient congestion control, several connection-oriented internetworking protocols[Top90, SJC92] have been developed. These protocols generally depend upon resource reservation to avoid congestion, however, reservation inherently wastes network resources. Besides, we cannot use such protocols on an internet which contains sub-networks that do not support resource reservation.

We propose a new design of a connection-oriented internetworking protocol which we call "CIP" (Connection-oriented Internet Protocol). CIP has two major characteristics:

- CIP enables intermediate gateways to participate in congestion control by credit-based hop-by-hop sliding window flow control in which congestion can be directly detected and avoided.
- Existing IP gateways can be included in CIP connection-paths between end-points. So CIP has the advantage that it can coexist with IP networks, and reuse functionalities incorporated with IP.

We present the implementation of CIP on Mach 3.0 as an application executed in the user address space, and evaluate it, focusing on efficient utilization of existing network resources.

In the face of recent progress found in increase of network bandwidth, overhead of CIP hop-by-hop flow control can become burdensome in the future. So we also incorporate mechanisms for rate control and buffer space pre-allocation into CIP connection management as provisions for future extensions of CIP.

Faculty of Science and Technology, KEIO UNIVERSITY

※ The contents of English summary should be written in about 300 words.

Contents

1 INTRODUCTION **1**
- 1.1 Traditional Congestion Control Strategies . . . 2
- 1.2 Our Proposal: CIP . . . 3
- 1.3 Organization of the Paper . . . 4

2 BACKGROUND and MOTIVATION **5**
- 2.1 Connectionless Internetworking Protocol . . . 6
  - 2.1.1 Internet Protocol (IP) . . . 6
  - 2.1.2 TCP/IP . . . 9
- 2.2 Connection-Oriented Internetworking Protocol . . . 12
  - 2.2.1 Internet Stream Protocol (ST-II) . . . 12
  - 2.2.2 The Xpress Transfer Protocol (XTP) . . . 16
- 2.3 Resource Reservation Protocol . . . 18
  - 2.3.1 The New Resource Reservation Protocol (RSVP) . . . 18

3 PROTOCOL DESIGN **20**
- 3.1 Advantages of Connection-Oriented Paradigm . . . 21
- 3.2 Store and Forward Delivery . . . 22
- 3.3 Ability to Coexist with IP . . . 24
- 3.4 Connection Management . . . 25
  - 3.4.1 Connection setup . . . 25
  - 3.4.2 Connection teardown . . . 30
  - 3.4.3 Exceptional cases . . . 30
- 3.5 Data Transfer . . . 31
  - 3.5.1 Flow control . . . 31
  - 3.5.2 Error control . . . 33
  - 3.5.3 Rate control . . . 35

4 IMPLEMENTATION **37**
- 4.1 Implementation Overview . . . 38
- 4.2 Fast Memory Management . . . 39
  - 4.2.1 Buffer . . . 39
- 4.3 Event Management . . . 41
  - 4.3.1 Timer . . . 41
  - 4.3.2 Queue . . . 42
- 4.4 Thread Layout Policy . . . 44
  - 4.4.1 Thread . . . 44
  - 4.4.2 Layout . . . 44
- 4.5 Device Input Output . . . 45
- 4.6 CIP Interface . . . 46
- 4.7 CIP Input Output . . . 47

4.7.1 Further abstraction of device abstraction . . . 47
4.7.2 IP input . . . 47
4.7.3 IP output . . . 48
4.7.4 Routing and address resolution . . . 49
4.7.5 CIP Thread . . . 49
4.8 Transport Protocol Input Output . . . 51
4.8.1 User Stream Protocol . . . 51
4.8.2 Multiplex and demultiplex . . . 52

5 EVALUATION **55**
5.1 Simple Performance . . . 57
5.2 Comparison of CIP gateway and IP gateway . . . 60
5.3 Rate Control . . . 62
5.4 Discussion . . . 70
5.4.1 Usability of CIP . . . 70
5.4.2 Overhead . . . 70
5.4.3 Fragmentation . . . 71
5.4.4 Out-of-band delivery . . . 71

6 Future Work **72**
6.1 Optimization of CIP . . . 73
6.1.1 Revision of congestion control strategy . . . 73
6.1.2 Separation of control messages and data . . . 73
6.2 Optimization of our implementation . . . 74
6.3 Simulation . . . 74
6.4 Resource management . . . 74

7 CONCLUTION **75**

Acknowledgments **77**

Bibliography **78**

Appendix **80**

List of Figures

1.1 Protocol Relationships 4

2.1 IP Datagram Format 8
2.2 IP Forwarding Mechanism 9
2.3 TCP Window Advertisement 10
2.4 TCP Congestion Window 11
2.5 TCP Retransmission Diagram 12
2.6 ST-II Connection Setup 14
2.7 XTP Rate Control 17
2.8 RSVP Resource Reservation Setup 19

3.1 An Internet 22
3.2 Store and Forward Delivery 23
3.3 CIP Option Format 24
3.4 CIP Connection Setup Procedure 27
3.5 CIP Connection Setup Diagram 29
3.6 CIP Sender Window 32
3.7 Slow-start 33
3.8 Congestion Avoidance Step 34
3.9 Retransmit Timer Back-off 35

4.1 Software Structure 39
4.2 Buffer Operation 40
4.3 Queue Operation 43
4.4 Thread Layout Policy 44
4.5 IP Receive Thread 48
4.6 CIP Thread Layout 50
4.7 CIP Gateway 50
4.8 USP Connection 51
4.9 CIP Multiplexing 54
4.10 CIP Demultiplexing 54

5.1 Our Experimental Internet 56
5.2 Simple Performance 58
5.3 Capability of Buffer Space 60
5.4 Time Sharing Effects 61
5.5 CIP Gateway 63
5.6 IP Gateway 64
5.7 CIP Gateway with Congestions 65
5.8 IP Gateway with Congestions 66
5.9 Rate Control 100KBps, 150KBps 68
5.10 Rate Control 192KBps, 256KBps 69

Chapter 1

INTRODUCTION

Congestion control is an integral part of networking. As for "internetworking", especially, an unsatisfactory design or a deficiency of mechanisms to avoid congestions leads to a sheer waste of available network resources. It is difficult to build such internetworking infrastructure as is envisioned as the information super highway without congestion control.

There are different levels of congestion control, that is, data-link layer level, network layer level, and transport layer level one. For example, Ethernet performs CSMA/CD (Carrier Sense Multiple Access / Collision Detect) at the data-link level. In this paper we focus on the network layer level congestion control and consider their mechanisms.

1.1 Traditional Congestion Control Strategies

"TCP/IP" is a representative internetworking protocol in the Internet. TCP/IP is a compound name of a protocol suite, TCP and IP. IP (Internet Protocol) provides a framework to deliver datagrams for transport protocols, and TCP (Transmission Control Protocol) performs transmission control on it. Many people have studied and improved them for a long time.

In recent years, it has come to light that IP is inadequate for such new type distributed applications as transport audio and visual continuous data in terms of congestion control[Top90, ZDE*93]. For efficient congestion control, several connection-oriented internetworking protocols [Top90, SJC92] have been developed and proved quite useful. These protocols generally depend upon resource reservation to avoid congestions. Naturally, however, these protocols did not have interconnectivity with IP at each switch.

Nevertheless, IP is still attractive, we think. The essence of the Internet architecture embodied in IP is summarized as follows: 1) point-to-point delivery; Datagram delivery from one IP address to another. 2) connectionless mode delivery; Intermediate systems can drop datagrams whenever their buffer overflows. Because of this simplicity, IP can be implemented on diverse platforms or operating systems, and the fact has enhanced interconnectivity of them on a worldwide scale. Meanwhile IP does not have any effective congestion control mechanisms in exchange for the simple architecture.

In general, transport protocols are held responsible for congestion control in the case that they reside on such connectionless protocols that do not equip effective congestion control mechanisms. In the Internet arena, people have traditionally attempted congestion control by using the TCP/IP protocol suite. It is true that TCP works well if we use it on a sub-network or across a few interconnected sub-networks. However, congestion control of TCP completely relies upon end-to-end flow control based on receiver's window advertisement, sender's congestion window independent of receiver's one, and estimation of smoothed round-trip time. Generally speaking, the more the connection-path is prolonged, the more uncertainness relevant to the end-to-end flow control increases. Hence, we cannot make the best use of TCP congestion

control mechanisms across multiple hopped internet where interconnecting effects appear.

On the other hand, when we use resource reservation protocols, there is usually unavoidable waste of network resources, especially network bandwidth, since we have to reserve the maximum amount of flow specification required to send or receive particular stream. Even if generation rate of continuous data such as video is stationary, they are usually compressed before we send them, therefore transfer rate varies below pre-allocated one as time goes on. In the end, the balance in hand will be lost in vain.

In contrast to this problem, TCP is adaptive to immediate available resources, that is, TCP can exhaust them efficiently within capability of allocated buffer and packet switch processing bandwidth. This is a notable feature which is left unconsidered among ideas of resource reservation from the beginning, for people have focused on the efficiency of not utilization of each reserved resource but reservation itself. Besides, there is a dilemma that it is regrettable to discard such unused resources, but when allowed maximum rate are delivered, we would be in trouble if they were preempted.

1.2 Our Proposal: CIP

In the face of these consideration, we propose a new connection-oriented internetworking protocol that we call "CIP" (Connection-oriented Internet Protocol) with two design goals described below:

- network layer level congestion control
  CIP enables intermediate gateways to participate in congestion control by credit-based hop-by-hop sliding window flow control in which congestion can be directly detected and avoided.

- ability to coexist with IP
  Existing IP gateways can be included in CIP connection-paths between end-points. So CIP has the advantage that it can coexist with IP networks, and reuse functionalities incorporated with IP.

Even though we cannot guarantee any amounts of transfer rate under the pre-specified upper bound, a connection-path along which datagrams are delivered is divided into some sections each of which is observed more exactly than before by both gateways surrounding the section. We think that by applying congestion control mechanisms to localized networks closely, we can advance efficient utilization of existing network resources.

We present the implementation of CIP on Mach 3.0, and evaluate it, focusing on efficient utilization of existing network resources. Protocol relationships are shown in Figure. 1.1. We also attempt to apply rate control to the CIP connection-paths through which CIP streams of

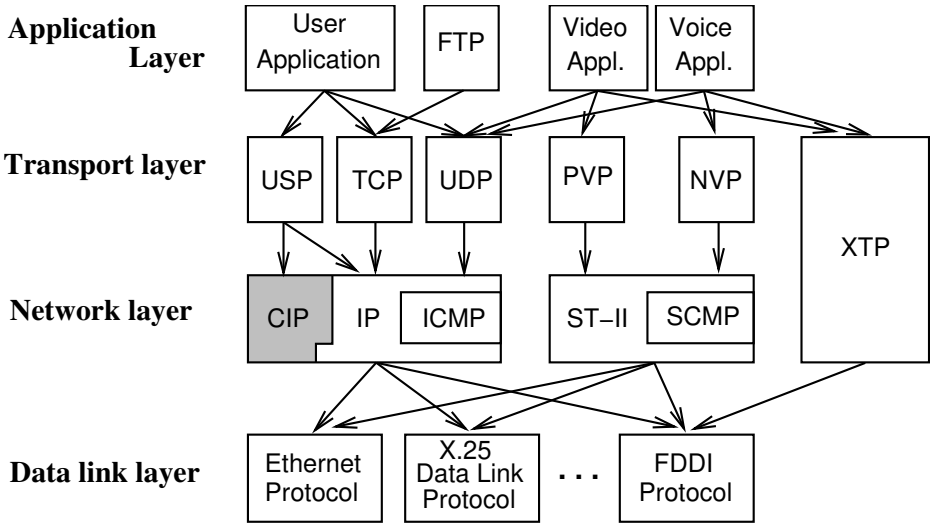


Figure. 1.1: Protocol Relationships

packets are delivered, including pre-allocation of buffers. Fortunately, there is a development environment suitable for protocol implementations, that is, Mach Micro Kernel where network protocols are decomposed from kernel implementations. Until the appearance of such environment, we had to struggle with monolithic kernels such as UNIX. Now we can develop new protocols more easily than before.

In the mean time, it is recalled that connection-oriented protocols have another role, that is, capability to support point-to-multi-point or multi-point-to-multi-point connection. But we dare choose point-to-point connection for clarity of our purpose for the present. Extension such as multicast support is left for future work.

1.3 Organization of the Paper

This paper first examines the evolution of network protocols as background. We then consider the essential characteristics of internetworking and design a new connection-oriented internetworking protocol, focusing on congestion control mechanisms and ability to coexist with IP. An implementation of this framework is next examined, including other components of the protocol stack. We follow with an evaluation of this implementation, examining both performance and usability. Finally, our conclusion is shown.

Chapter 2

BACKGROUND and MOTIVATION

In this chapter, we examine the evolution of network protocols, focusing on congestion control strategies. In general, there are two types of internetworking protocols, that is, connectionless internetworking protocols and connection-oriented ones. They typically have following congestion avoidance mechanisms respectively:

- Connectionless internetworking protocols
  - Discarding pacekts
    Overloaded intermediate systems only discard incoming packets.
  - Source quench
    Overloaded intervening gateways request to reduce injecting rate of packets for the source host injecting packets into the network in high rate which makes their buffer overrun.
- Connection-oriented internetworking protocols
  - Flow control
    Mismatches between a transmitter's ability to deliver data to the network service and the receiving peer's ability to process and buffer that data can severely degrade communication efficiency. For sake of preserving receiver's buffers from overrunning, the sender and the receiver exchange control messages which restrain injecting rate of packets at the sender.
  - Rate control
    In order to preserve both the end-points and intermediate nodes from buffer overrunning, another constraint, that is, rate control is applied to the connection. Rate control restricts transfer rate of packets at each node.
  - Resource reservation
    To avoid congestion, some network protocols reserve network resources required to deliver particular streams of packets in advance.

Hereafter we review respective internetworking protocols, paying attention to above conventional congestion control mechanisms.

2.1 Connectionless Internetworking Protocol

2.1.1 Internet Protocol (IP)

Internet Protocol (IP) was developed for the Internet, which is a large scale packet switching network today, with financial support of DARPA[1]. IP corresponds to the network layer in the

[1] Defense Advanced Reserch Projects Agancy.

OSI Reference Model[Tan89] and provides point-to-point datagram delivery that is based on a hop-by-hop mechanism where each message is relayed by intermediate gateways one after another from an origin host to a target host. IP encapsulates the detail of the internetworking (e.g., its forwarding mechanism, difference of data-link protocols or hardware properties, and so on) and hide it from transport layers.

The essence of Internet architecture embodied in IP is summarized as follows:

- point-to-point delivery
  Datagram delivery from one IP address to another. A major advantage of the datagram approach to internetworking is that it can be used over sub-networks that do not use virtual circuits inside. Many LANs, mobile networks (e.g., aircraft and naval fleets), and even some WANs fall into this category. When an internet includes one of these, serious problems occur if the internetworking strategy is based on virtural circuits.

- connectionless mode delivery
  Intermediate systems can drop datagrams whenever their buffer overflows. This "drop" plays a role as congestion control in IP, so IP potentially adapts congestion, even though IP also potentially cause it. In addition, IP has robustness in the face of gateway failures.

Because of this simplicity, IP can be implemented on diverse platforms or operating systems, and the fact has enhanced interconnectivity of them on a worldwide scale. Meanwhile IP does not have any effective congestion control mechanisms in exchange for the simple architecture.

In fact, a reporting mechanism that is used to inform the source host of the discard of packets is prepared as ICMP (Internet Control Message Protocol) that manages things relevant to IP, exchanging control messages autonomously among gateways and hosts. In particular, such discard message which requires the source to restrict injection of packets is called as "source quench". Source quench messages perform rudimentary flow control. The host received the source quench message throttles its injection rate until the requests does not arrive any longer.

There are several alternatives in the face of issuing source quench requests. For example, one gateway can issue them for all source hosts from which the dropped datagram traveled, while another sophisticated gateway does only for particular hosts which are relatively affective to the congestion, for example, the host that is judged injecting packets into the network in the most high transfer rate through measurements at the overloaded gateway. But there are few such sophisticated implementations in general; besides some IP gateways say nothing while discarding packets.

Here we review fundamental mechanisms of IP datagram delivery briefly for later implementation.

As every data-link frame usually consists of a pair of data-link header and data, so every IP datagram consists of a pair of IP header and data. An IP header contains several descriptors,

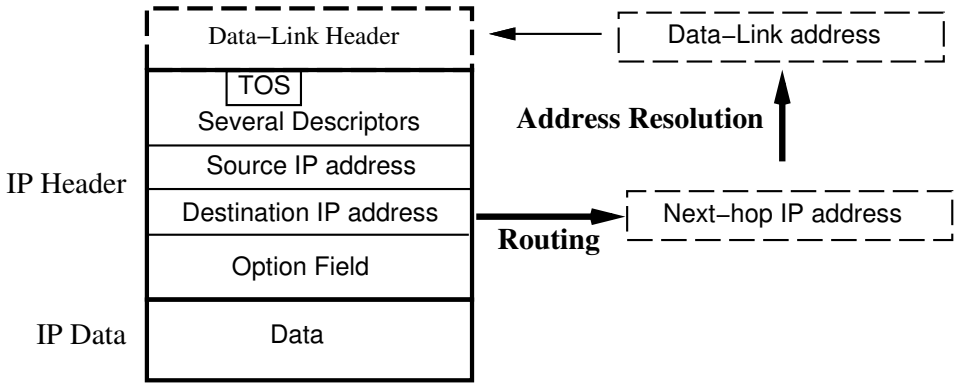


Figure. 2.1: IP Datagram Format

in particular, a source IP address, a destination one, and a type of service (TOS) field (shown as Figure. 2.1.)

TOS indicates the required service for the datagram as below:

- D bit: indicates "low delay".
- T bit: indicates "high throughput".
- R bit: indicates "high reliability".
- Precedence: is represented by 3bit number.

TOS enables hosts or gateways to choose the best route for each datagram according to the service type. For example, when a gateway is connected to both a broad-band satellite network which has generally high delay and a narrow-band private line which has low delay, the gateway can choose the satellite network for datagrams signed T bit, while it can choose the private line for datagrams signed D bit, which are probably used for terminal programs such as telnet.

In practice, this IP header is used as below.

When sending a datagram, we invoke a "routing" function in the first place to find a next-hop's IP address, or possibly the target itself, and convert the result into a data-link layer address such as Ethernet address. This conversion is called as "address resolution". Then we send the datagram to the temporary data-link address, leaving the IP header as it is.

When receiving the datagram, first of all, we examine the destination IP address and see if it is destined for us. If so, we demultiplex it to upper protocols, otherwise we invoke a routing function and forward the datagram to a new next-hop in the same way as the first delivery. And above is repeated until the datagram reaches the destination.

Like this, the IP forwarding (or delivery) mechanism is principally composed of routing and address resolution (shown as Figure. 2.1, 2.2.) and is quite simple.

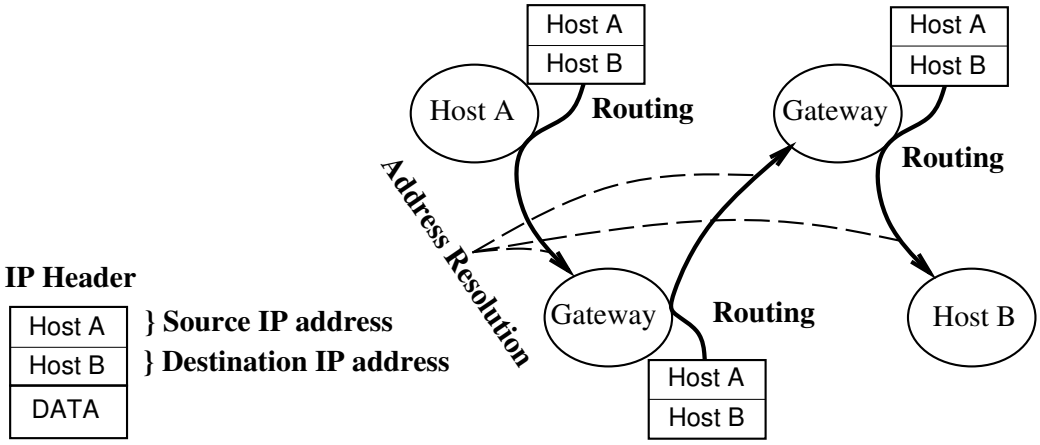


Figure. 2.2: IP Forwarding Mechanism

But when it comes to address resolution and routing themselves, they are not very or at all simple respectively. Address resolution is encapsulated as ARP (address resolution protocol) for each data-link protocol that IP uses, for example, the address resolution protocol of Ethernet is defined in [Plu82] completely. As for routing, there are various solutions handling it from fixed routing tables to autonomous gateway protocols. A fixed routing table is adequate for a small network where the number of hosts is limited and they are statically arranged. But it is not suitable for large internets in which network topologies often change, where it is desirable that gateways manage routing informations autonomously.

We do not refer to routing itself closely, because it is not our subject in this paper. But we take notice of OSPF Version 2 (Open Shortest Path First) specified in [Moy94]. OSPF gives us the best route, performing the routing function based on the destination IP address and the type of service (TOS) field in the IP header. OSPF is a dynamic routing protocol. It quickly detects topological changes (such as gateway interface failures) in the internet and calculates new loop-free routes after a period of convergence. This period of convergence is short and involves a minimum of routing traffic. OSPF calculates separate routes for each type of service. When several equal-cost routes to a destination exist, traffic is distributed equally among them. The cost of a route is described by a single dimensionless metric.

2.1.2 TCP/IP

In general, transport protocols are held responsible for congestion control in the case that they reside on such connectionless protocols that do not equip effective congestion control mechanisms. In the Internet arena, people have traditionally attempted congestion control by using the TCP/IP protocol suite, where TCP (Transmission Control Protocol) corresponds to

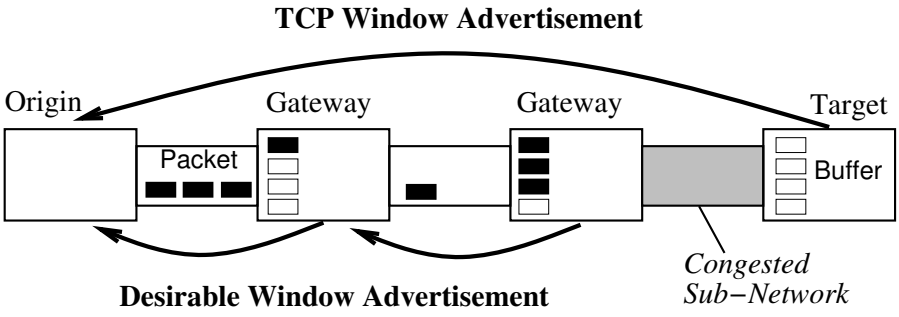


Figure. 2.3: TCP Window Advertisement

the transport layer in the OSI Reference Model. TCP provides sequenced error-free, that is, reliable streams of packets, using IP by a way usually referred to as best-effort delivery that tries to send a datagram over and over unless it reaches a target, which stems from the inherent characteristics of IP's unreliable circuit.

Apparently, TCP does not seem to work well, since TCP cannot see the detail of the based networks. Nobody tells TCP how much bandwidth is available or whether the intermediate networks are congested or not. How does TCP avoid congestions so successfully? The answer is quite simple. TCP only employs end-to-end credit-based sliding window flow control as a method of congestion control. To our surprise, this method works effectively for various sub-networks such as fiber optic networks, satellite networks, and packet radio networks.

The key components of TCP's flow control are following:

- Receiver's window advertisement

  TCP employs credit-based sliding window flow control. Receiver must advertise the current available buffer size, so sender can restrain the flow in response to it (as shown in Figure. 2.3). In other words, this mechanism prevents receiver's buffer from overflowing.

  But this window advertisement has a serious problem. When there is some congested sub-networks between an origin and a target, without no additional strategy, TCP window advertisement would make these congestion worse, that is, the gateway's buffer at the entrance point of the congested network might be filled up, nevertheless, the target went on advertising its probably empty window, consequently the congestion might get worse. In reality, TCP avoids such situations by using the congestion window described below, but we think it is desirable that intermediate systems advertise their own windows directly by themselves, if possible.

- Sender's congestion window

  Sender has another window that we call a congestion window, which is independent of

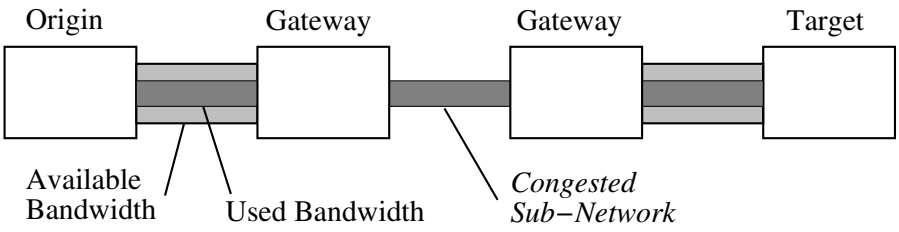


Figure. 2.4: TCP Congestion Window

receiver's window advertisement and always takes precedence over it. Whenever datagrams are lost, sender reduces the congestion window as a symptom of a congestion and succeedingly increases it gradually, seeking an adequate point of the flow to ease the congestion (as shown in Figure. 2.4.)

Generally speaking, the more the connection-path is prolonged, the more uncertainness relevant to it increases. Nevertheless, TCP flow control must depend upon such indirect congestion detection and avoidance mechanism.

- Estimation of smoothed round-trip time
  TCP measures round-trip time about each datagram and calculates their mean and variation[2]. Whenever one datagrams's round-trip time exceeds the estimated one, TCP infers that it was lost and retransmit it (as shown in Figure. 2.5.)

  But there are some problems. First, a TCP round-trip time across the Internet has generally large variation, which accounts for diminished throughputs, for needless retransmissions or contrary extra delay of them may occur. Second, TCP always judges the time out at the origin, so when an intervening gateway fails to forward a packet to the next-hop anyhow, it takes longer time than judges at the front of the failure. Third, TCP retransmission is always performed by the origin. This results in duplicate transmissions between the origin and the failure, and these transmissions may possibly cause another congestion in that region.

It is true that TCP works well if we use it on a sub-network or across a few interconnected sub-networks. But we cannot make the best use of TCP congestion control mechanisms across multiple hopped internets where interconnecting effects appear, namely, where receiver's window advertisement loses its meaning, and in order to preserve intervening systems from congestion we have to completely depend upon sender's sole congestion window which converges on the equilibrium point while oscillating.

[2] The practice is shown in Section. 3.5.2.

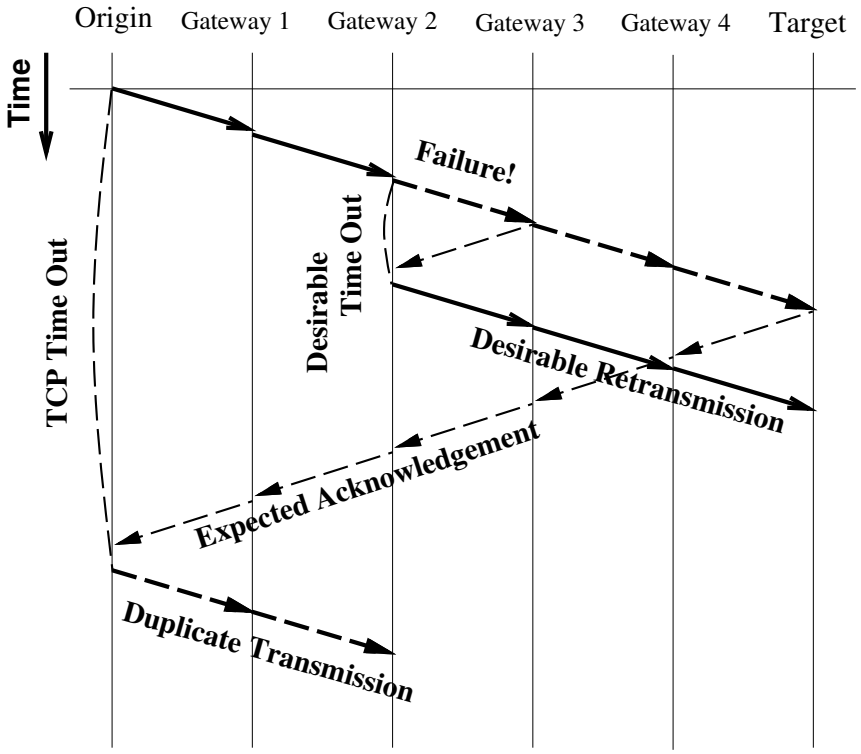


Figure. 2.5: TCP Retransmission Diagram

In this diagram, gateway2 failed to forward a datagram to the next-hop, gateway3. To apply TCP time out to this failure results in duplicate transmission between origin and gateway2.

It is natural that we hope intermediate systems to join us in congestion control. Note that because of ability to coexist with IP, we cannot make all of them so, that is, there are still old gateways do not understand our new protocol. Therefore we can recognize as follows: a connection-path along which datagrams are delivered is divided into some sections each of which is observed more exactly than before by both gateways surrounding it. By applying congestion control mechanisms to localized networks closely, we can advance efficient utilization of existing network resources.

2.2 Connection-Oriented Internetworking Protocol

2.2.1 Internet Stream Protocol (ST-II)

In recent years, it has come to light that IP is inadequate for such new type distributed applications as transport audio and visual continuous data in terms of congestion control[Top90, ZDE*93]. For efficient congestion control, several connection-oriented internetworking protocols

[Top90, SJC92] have been developed and proved quite useful. These protocols generally depend upon resource reservation to avoid congestions. Naturally, however, these protocols did not have interconnectivity with IP at each switch.

As an example, we examine Internet Stream Protocol, Version 2 (ST-II), which is an experimental network protocol whose specification is described in [Top90].

ST-II has been developed to support efficient delivery of streams of packets to either single or multiple destinations in applications requiring guaranteed data rates and controlled delay characteristics. The motivation for the original protocol, ST, was that IP did not provide the delay and data rate characteristics necessary to support voice applications.

ST-II is an internet protocol at the same layer as IP. ST-II differs from IP in that IP, as originally envisioned, did not require routers (or intermediate systems) to maintain state information describing the streams of packets flowing through them. ST-II incorporates the concept of streams across an internet. Every intervening ST entity maintain state information for each stream that passes through it. The stream state includes forwarding information, including multicast support for efficiency, and resources information, which allows network or link bandwidth and queues to be assigned to a specific stream. This pre-allocation of resources allows data packets to be forwarded with low delay, low overhead, and a low probability of loss due to congestion. The characteristics of a stream, such as the number and location of the endpoints, and the bandwidth required, may be modified during the lifetime of the stream. This allows ST-II to give a real time application the guaranteed and predictable communication characteristics it requires, and is a good vehicle to support an application whose communications requirements are relatively predictable.

It is notable that ST-II introduced the concept of streams into IP level, although it had already been thought in some protocols such as CCITT X.25[3]. Indeed, ST-II became the starting point of internet stream protocols in the Internet. Our new protocol, CIP also is based on ST-II, especially in its connection management. Here we review ST-II fundamental connection management mechanisms briefly.

Connection setup

On ST-II, we must setup a connection before data transfer. First of all, we examine a target list which may include several targets because of multicast support, and invoke routing function to find next-hops' IP addresses that are classified for each district. Then we convert the result into a data-link layer address respectively and send a connect message containing some descriptors (Hop Identifier, Virtual Link Identifier, etc.), which are used to quickly dispatch data to be forwarded or control messages, with a corresponding target list and required flowspec

[3] X.25 is a CCITT standard protocol that defines an interface between packet switching nodes and terminals. But there is no specification of the internal networks.

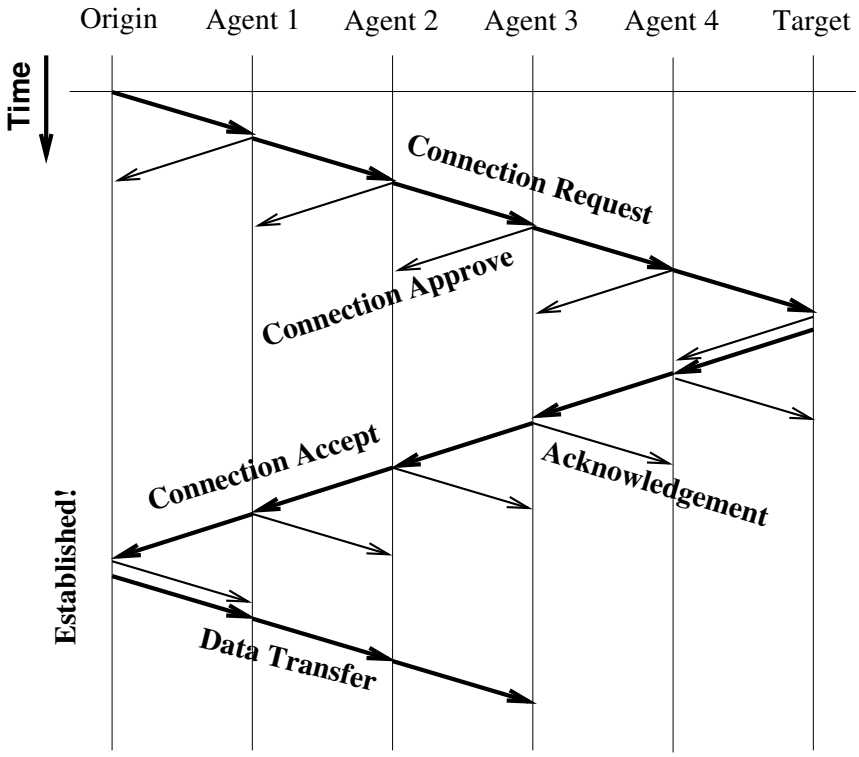


Figure. 2.6: ST-II Connection Setup

(flow specification) to the temporary data-link address.

The peer ST agent, which is ST-II terminology indicates intermediate systems, that received the connection request sees if it can acquire enough resources to satisfy the required flowspec. If can, it sends back an approve message, otherwise a refuse one. If the ST agent is not the target, it propagates the connection request similarly as the first message, otherwise it sends back an accept message to its peer. When this message is eventually propagated back to the origin, the connection is established fully (as shown in Figure.2.6.)

Data transfer

Once connection is established, continuous data can be delivered through it. Because they pass along pre-allocated resources proper to the connection, there is no necessity for additional congestion control mechanisms no longer. In practice, ST-II has no such mechanisms, besides, it does not also issue any acknowledgement about datagrams of the stream. ST-II does neither detect packet loss nor perform sequencing and retransmissions, so ST-II connections have no reliability, only to depend upon resource reservation.

It is recalled that ST-II does not have interconnectivity with IP, that is, ST agents must be located throughout a connection-path of a stream, and network resources there also must be

ready to be reserved entirely. This problem is pointed out, too, in [Top90], where they describe that few networks in the Internet currently offer resource reservation, and none that they know of offer reservation of all the resources that they specified. Only the Terrestrial Wideband Network (TWBNet) and the Atlantic Satellite Network (SATNET) offer(ed) bandwidth reservation. No network provides for the reservation of packet switch processing bandwidth or buffer space.

In the future such environment may be turned into reality, but we suppose it is difficult to realize at once throughout the Internet and we must not ignore ability to coexist with IP for the present.

Modifying an existing stream

Some applications may wish to change the parameters of a stream after it has been created. For that purpose, ST-II has a changing mechanism that issues change messages, which are structured and processed similarly to connect messages. Possible changes include adding or deleting targets and changing the flow specification.

Stream teardown

A stream is usually terminated by the origin when it has not further data to send, but may also be partially torndown by the individual targets. These cases are processed as modifying the stream.

A stream is also torndown if the application should terminate abnormally. Processing in this case is identical to the previous descriptions except that the appropriate reason code is different.

When all targets have left a stream, the origin notifies the application of that fact, and the application then is responsible for terminating the stream. Note, however, that the application may decide to add a target(s) to the stream instead of terminating it.

IP encapsulation of ST-II

Although we pointed out that ST-II has no interconnectivity with IP, in fact, ST-II offers it in a sense. ST packets may be encapsulated in IP to allow them to pass through routers that do not support ST-II. Authors of ST-II said as following[Top90]: Of course, ST resource management is precluded over such a path, and packet overhead is increased by encapsulation, but if the performance is reasonably predictable this may be better than not communicating at all.

But there is a serious problem herein. As described previously, ST-II provides no congestion control mechanisms except for pre-allocation of network resources. Since ST-II is completely

based on resource reservation, it is fragile against nonfulfilment of the premise. That is, there is possibility not only that encapsulated ST packets are lost somewhere, but also that the ST packets cause another congestion. So we recognize ST-II does not have interconnectivity with IP.

2.2.2 The Xpress Transfer Protocol (XTP)

XTP (The Xpress Transfer Protocol)[SJC92] is a connection-oriented internetworking protocol which contains an integration of the network layer and the transport layer in the OSI Reference Model. XTP's "eXpress" is derived from its radically high-band transport capability which is mainly based on fiber optic networks such as FDDI (Fiber Distributed Data Interface). XTP provides with reliable connections which have several enhancements such as:

- very high data throughput capability for operation over high speed transmission facilities (e.g., ATM, FDDI, and Broadband-ISDN).
- end-to-end flow control.
- hop-by-hop rate control.
- selectable error control procedures. (consequently, XTP connections' reliability is optional.)
- multicast operation to support multipeer applications.
- quality of service selection and management such as priorities handling.
- out-of-band signaling and synchronization.

It is notable that XTP addresses hop-by-hop rate control as a congestion avoidance mechanism. Since the rate of packet injection at the sending end-node should be constrained by the rate of packet processing at the slowest intermediate node along the path between the end-nodes, there must be some mechanisms for communicating this "weak-link" rate back to the sending end-node. Unfortunately, most network layer protocols do not have an explicit facility for either informing the sending end-node or controlling the rate of injection even if that rate could be known. In this respect, congestion control through controlling packet injection rates is an end-to-end issue, since it involves information from the entire path; on the other hand, it is really the processing rates at each individual switching node that causes the congestion in the first place, and so in some sense congestion control is a hop-by-hop issue. Certainly, cooperation between end-to-end services and the hop-by-hop services supporting them must take place, yet the boundary between the transport and network layers often prevents this cooperation.

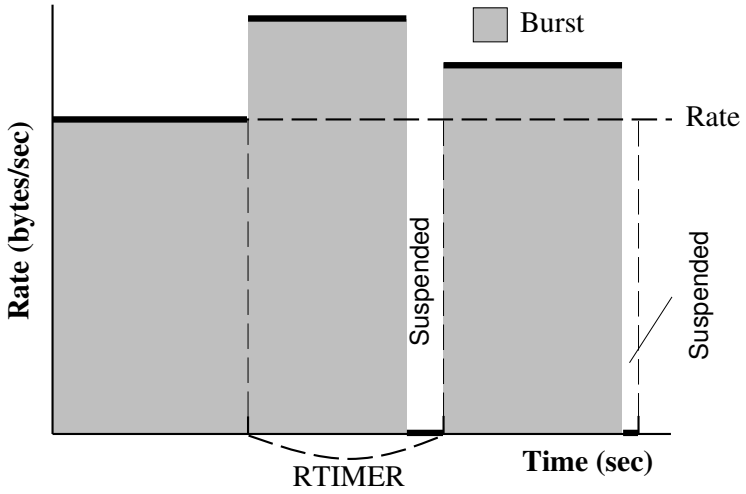


Figure. 2.7: XTP Rate Control

The initiating context of a XTP connection begins with default values for "rate" and "burst" parameters. The rate value specifies in bytes per second the (supposed) maximum rate at which data can be consumed by the receiver. The burst specifies the (supposed) maximum number of bytes which can be consumed in one burst of packets, that is, packets sent in rapid succession. These values are "supposed" since they are defaults and thus are not based on actual feedback from the receiver. The XTP Definition defines a countdown timer called the RTIMER (for refresh timer) for each connection, which will mark regular periods over the lifetime of the connection. The value placed in the RTIMER is the quantity *burst* divided by *rate* since this quotient gives us the period of time during which *burst* number of bytes can be transmitted. An association variable, "credit", is loaded with the value in burst each time the RTIMER is rest. As packets are transmitted, credit is decremented by the amount of data sent. When the value in credit reaches zero, the transmission of data packets is suspended until RTIMER expires. At this time the RTIMER is refreshed, the credit is renewed, and transmission of data packets is once again enabled (as shown in Figure. 2.7.)

Each one of the intermediate nodes along the path between the endpoints of the connection may participate in the rate control procedure. Switches are permitted to reduce the value of the *rate* and *burst* fields of any control packets (which are used to manage things relevant to XTP) passing through them. The rate of data generation at the transmitter is thus the minimum acceptable rate of all of the nodes along the path.

In the mean time, XTP does not dictate any method to make reservation for required network resources; besides, it does neither any method to estimate the amounts of network bandwidth used in rate control. So we think such estimation method should be addressed as soon as

possible, and we can include a rate control mechanism just like XTP to our protocol easily.

2.3 Resource Reservation Protocol

Concerning resource reservation, there is usually unavoidable waste of network resources, especially network bandwidth, since we have to reserve maximum amounts of flow specification required to send or receive particular stream. Even if generation rate of continuous data such as video is stationary, they are usually compressed before we send them, therefore transfer rate varies below pre-allocated one as time goes on. In the end, the balance in hand will be lost in vain.

In contrast to this problem, TCP is adaptive to immediate available resources, that is, TCP can exhaust them efficiently within capability of allocated buffer and packet switch processing bandwidth. This is a notable feature which is left unconsidered among ideas of resource reservation from the beginning, since people have focused on the efficiency of not utilization of each reserved resource but reservation itself. For example, RSVP (a new resource ReSerVation Protocol)[ZDE*93] provides a new approach to efficient resource reservation.

2.3.1 The New Resource Reservation Protocol (RSVP)

RSVP has an unique feature against ST-II. While ST-II employs sender-oriented reservation style described previously, RSVP proposes receiver-oriented reservation, which enables receivers to specify their application needs, so the aggregate resources reserved for a multicast group can more accurately reflect the resources actually needed by that group.

It is important for us to study these reservation protocols, in particular their capability of multicast support, although we chose point-to-point connection for our protocol for clarity.

Here we survey the RSVP reservation mechanism briefly. The example network has five hosts connected by seven point-to-point links and three switches (Figure. 2.8) We consider the case where H2, H3, H4, and H5 are receivers (i.e., members of the multicast group), and H1, H4, and H5 are sources. Each switch uses the path states to maintain a table of incoming and outgoing interfaces for each multicast group. Assume that S1 has received path messages from all of the sources but no reservations have yet been made. Thus, S1's path state contains the following entry:

```
S1: L2(H1,H1|H4,S2|H5,S2), L6(H1,H1)
```

The notation indicates "Switch: Link(Outgoing Link, Previous Hop | Outgoing Link, Previous Hop | ...)".

Now assume that H2 sends a reservation message denoted R2(B,H4), that is, H2 wants to receive packets only from source H4 and is reserving an amount B, sufficient for one source.

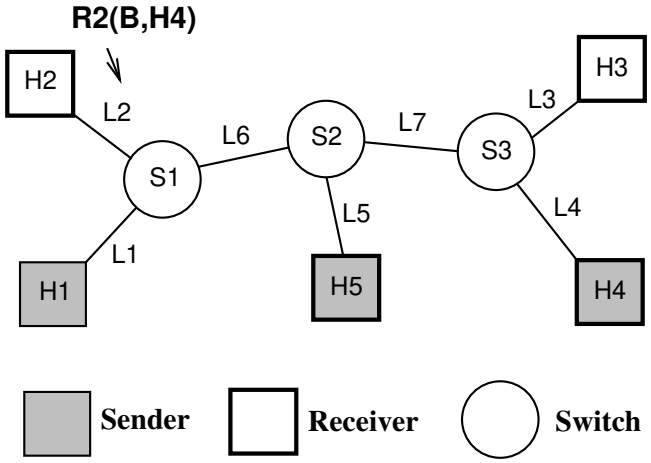


Figure. 2.8: RSVP Resource Reservation Setup

R2(B,H4) reaches S1 via the L2 interface. S1 finds that H4 is indeed one of the sources it has heard, and that the packets from H4 come from S2. S1 reserves bandwidth B over L2, and forwards R2(B,H4) over L6 to S2.

S2's path state contains the following entries:

```
S2: L5(H1,S1|H4,S3), L6(H4,S3|H5,H5), L7(H1,S1|H5,H5)
```

When S2 receives R2(B,H4), it reserves B over L6, and then forward the message R2(B,H4) to S3, which is the previous hop toward H4.

S3's path state then has its entries changed to:

```
S3: L3(H1,S2|H4,H4|H5,S2), L4(H1,S2|H5,S2), L7(H4,H4)
```

Upon receiving R2(B,H4), S3 reserves B over L7, and forward the message to H4. When the message reaches H4, a pipe of B has been reserved from H4 to H2. This describes the reservation events surrounding the reservation request R2(B,H4).

Suppose that sometime afterwards, H5 sends the reservation message R5(2B,*), where * indicates a request for dynamic-filter reservation. When S2 receives this reservation message, it reserves 2B over L5 (at least 2 sources can go that direcetion) for H5, and forwards the reservation message R5(2B,*) over L6 and L7.

When S1 receives R5(2B,*), it finds out that there is only one source going out L6. It therefore reserves an amount B over L6 for R5 and then passes the reservation request on to H1. When S3 receives R5(2B,*), it finds out that there is only one source going out L7 and has a fixed-filter reservation already (i.e., R2(B,H4)).

Like above, RSVP eliminates dupulicate reservations and consequently performs efficient resource reservation.

Chapter 3

PROTOCOL DESIGN

We describe a new design of a connection-oriented internetworking protocol to perform network layer level congestion control with store and forward delivery, retaining ability to coexist with IP. Hereafter we explain the detail of the new experimental protocol that we call CIP (Connection-oriented Internet Protocol).

CIP is a network layer protocol related with IP closely and provides upper protocols (i.e., transport layer protocols) with simplex (or, one direction) streams of packets, which is not an omission against duplex ones, for we intend an extension of CIP toward multicast support, where every connection-path topology generally composes a tree, that is, directed graph. Besides, even though a local IP path is usually duplex[1], a connection-path is not always suitable for inverted data transfer[2].

Meanwhile, CIP streams are unreliable like IP in the strict sence of the word. Although CIP usually keeps the streams as reliable as IP, namely, CIP always provides sequenced data streams where each datagram's header checksum is examined by IP, but CIP cannot guarantee the reliability in exceptional cases such as gateway failures, for there is traditionally an underlying philosophy in the Internet that the responsibility for ensuring that forwarded datagrams are correct is ultimately left for end-to-end entities, even if datagrams are delivered anyway. In the final analysis, we describe that CIP provides with "semireliable" streams of packets.

CIP is different from IP in that CIP provides transport protocols with service access points proper to each stream. So upper protocols have to get them from CIP before they send datagrams. Once a connection is established, user data are sent through the service access point and received by the target CIP agent and passed to the peer upper protocol. Note that CIP does not allow a stream to deliver datagrams which belong to different upper protocols.

3.1 Advantages of Connection-Oriented Paradigm

Here we explain the reason we aim at connection-oriented internetworking protocols, for there is some doubt as to significance of connection managements such as estimation of smoothed round-trip time. They say that because congestion control is interested in a section of an internet through which particular streams flow, it is desirable to estimate a round-trip time not for an individual stream but for aggregate streams from a gateway to the next-hop in terms of overheads.

Aggregation of each estimation of round-trip time is equivalent to aggregation of each sliding window. Now suppose a small internet as Figure.3.1, where A, B, C, G are CIP agents and they are interconnected via G. The link between G and C has narrow bandwidth, while the link between G and B has broad bandwidth. There are only two connections, one is C1 routed

[1] So we can return acknowledgement along it reversely. But CIP also allow acknowledgements to be delivered along another path.

[2] In particular, packet radio networks have such characteristics.

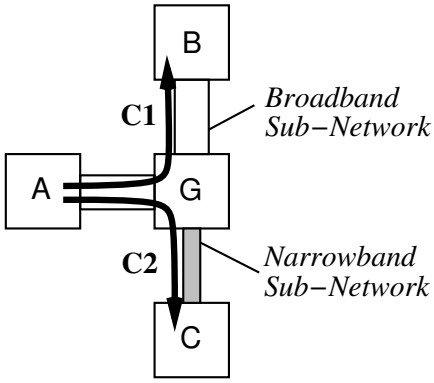


Figure. 3.1: An Internet

from A to B through G, another is C2 routed from A to C through G, respectively.

According to an idea of aggregation, we should manage one sliding window for these two streams. So packets belong to C1 cannot skip anyhow packets belong to C2 until they are forwarded successfully, despite of potentially available bandwidth given to C1.

A simple idea occurs, let C1 packets skip C2 packets, it is easy! But is it indeed correct? If one can distinguish C1 packets from C2 packets, he probably measures two round-trip time each of which belongs to C1 or C2. Furthermore, If one can manage one sliding window by two different kinds of manner, it is not too much to say that he has two different sliding windows.

Maybe another idea occurs, aggregation per each destination IP address is perhaps preferable. But this is also incorrect. Suppose the case that there are two connections on the same route from A to B through G, and two application such as telent and ftp which have different properties from each other are running at B. Such aggregation causes similar problems pointed out above.

Ultimately, we cannot aggregate different connections which have different properties from one another. Conversely, connection-oriented protocols have opportunities to make the best use of its advantages to manage each stream quite closely. For example, OSPF enables us to choose different routes for the same couple of origins and targets according as the TOS field.

3.2 Store and Forward Delivery

As we described previously, we employ the store and forward delivery mechanism with credit-based hop-by-hop sliding window flow control in our protocol for following reasons:

- Make intermediate gateways participate in congestion control.
  We employ the store and forward delivery mechanism, which enables intervening systems

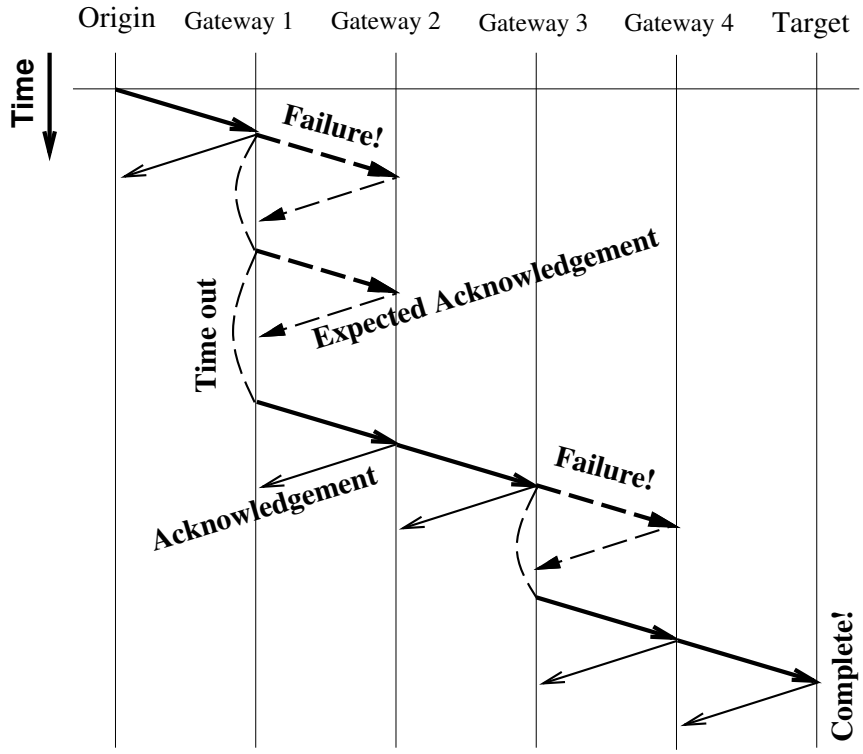


Figure. 3.2: Store and Forward Delivery

to participate in congestion control without reservation of network bandwidth by flow control mechanisms like TCP which is able to fit available resources.

- Reduce needless duplicate transmissions.
  When an intervening gateway fails to forward a packet to the next-hop anyhow, retransmissions at an origin result in duplicate transmissions between the origin and the failure, and it may possibly cause another congestion in that region.

The store and forward delivery itself is not a novel mechanism at all. Indeed, there were some packet radio networks which employed a network layer protocol known as NET/ROM[NET87] that performs the store and forward delivery likewise on AX.25 data-link protocol[3]. NET/ROM was developed for such packet radio networks as have high probability of packet loss. In those days, people focused on reducing needless duplicate transmissions rather than other components of congestion control.

But the store and forward delivery with credit-based hop-by-hop sliding window flow control has other advantages, namely, capability for intermediate systems to perform congestion control like below:

[3] AX.25 is a subset of X.25 data-link protocol, not including X.25 network protocol.

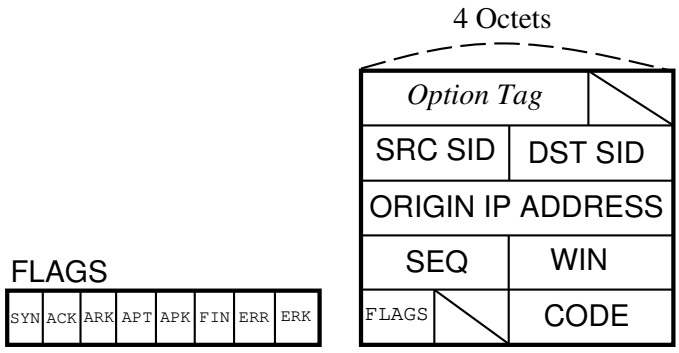


Figure. 3.3: CIP Option Format

- Window advertisement as a indicator of congestions
  Now that intervening gateways advertise their own window size, when there is a congested network on a connection-path, window advertisement of the gateway at the front of the congestion behaves as a kind of a signal that directly tells backward-hops there is congested network beyond the gateway, which propagetes back and restrains the flow through the connection.

- Estimation of smoothed round-trip time
  A round-trip time across the Internet has generally large variation, which accounts for diminished throughputs, for needless retransmissions or contrary extra delay of them may occur. By observing a sub-network closely between one gateway and another, we may get a more accurate variation, which can improve not only efficiency of network utilization but also possibly lead to unexpected high throughputs.

3.3 Ability to Coexist with IP

There are two reasons we intend to preserve ability to coexist with IP. First, it has advantages to coexist with IP and reuse property of IP. As we have emphasized up to here, prominent interconnectivity of IP is quite attractive. Second, when we use an IP network that is based on a connection-oriented data-link interface such as ATM, we can deliver CIP datagrams as IP ones on purpose, since there is no necessary to duplicate connections on the same sub-networks.

In practice, we reuse existing IP header format and only add new IP option fields which contains some descriptors required to specify the stream (as shown in Figure. 3.3). This allows intermediate gateways which do not understand CIP to skip the new option and forward a CIP datagram as if it were an IP one.

Descriptors have following meanings respectively:

- SRC SID, DST SID (Source Stream ID, Destination Stream ID);
  A number that identifies the stream and is unique at the local source or destination of the datagram.
- ORIGIN IP ADDRESS (The Origin IP address);
  Because the source IP address of IP header is rewritten (as shown in Figure. 3.4.), we must have this field to inform others of the real source (i.e., origin) IP address.
- SEQ (Sequence number);
  In case of sending user data, it means the sequence number identifies each datagram. Otherwise, it means the expected next sequence number in the cumulative positive acknowledgement policy.
- WIN (Window size);
  In case of issuing acknowledgement, it means receiver's window advertisement. And we also experimentally use this field for several purpose, for example, to indicate pre-allocating buffer size in the face of connection setup.
- FLAGS;
  A set of several flags:
  - SYN; Connection setup request.
  - ACK; Cumulative positive acknowledgement (used in several aspects).
  - ARK; Acknowledgement of the connection approvement.
  - APT; Connection accept message.
  - APK; Acknowledgement of the APT.
  - FIN; Connection teardown request.
  - ERR; Indication of an error (ERR validates the next CODE field).
  - ERK; Acknowledgement of the ERR.
- CODE; A number that represents a reason of the error.

We manage our protocol by such a simple IP option, leaving the original IP header as it is.

3.4 Connection Management

3.4.1 Connection setup

Connection setup of CIP resembles that of ST-II. Difference is following:

- CIP does not have the HID (Hop ID) negotiation process for efficient multicast like ST-II because of the limitation that CIP only provides point-to-point connection.
- CIP reserves no amount of network bandwidth but buffer space experimentally. While, it is currently under way that an universal metric for criteria of pre-allocating packet switch processing bandwidth.
- CIP has ability to coexist with IP.

Major differences between CIP and other connection-oriented internetworking protocols appear in the phase of data transfer.

Note that we allow upper protocols to include their own setup data into the first CIP datagram when setting-up a CIP connection. For example, USP (User Stream Protocol)[4] constructs an USP header as their abstraction including USP ports, and passes it to CIP. Of course, CIP does not understand the meanings of those data at all, only to deliver them.

Connection request

In the first place, an origin must send a connection request to a target. Like ST-II, we look up the target IP address and invoke routing function to find next-hop's IP address. Then we convert the result into a data-link layer address and send a connect message containing following descriptors to the temporary data-link address:

- Source Stream ID
  SRC SID is non-zero unique number at the source host or gateways and used to quickly dispatch data to be forwarded or control messages.
- Destination Stream ID
  So far, DST SID is yet to be identified. So we must fill this field with zero which is never used as DST SID for established streams.
- Origin IP address
  Target IP address is always known via an IP header contained in the first connection request, but Origin IP address is rewritten in the process of connection setup (as shown in Figure. 3.4.), so we include this field into CIP header.[5]
- SEQ
  SEQ indicates a series of sequence number. The first sequence number is choosen at random.

[4] As UDP (User Datagram Protocol) provides us with simple abstraction of IP, so USP gives us abstraction of CIP similarly. We describe it afterward.

[5] We describe IP header including CIP option as CIP header.

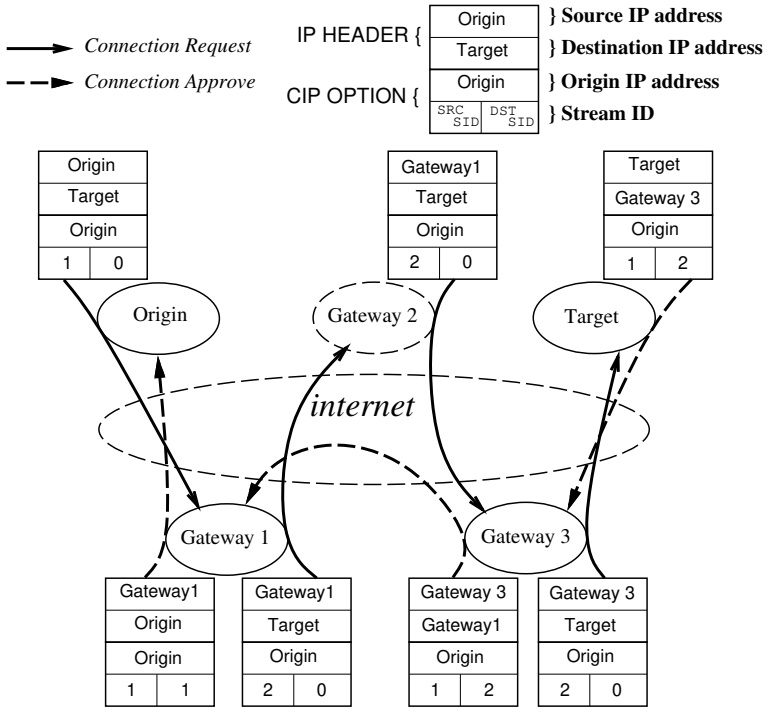


Figure. 3.4: CIP Connection Setup Procedure

In the first place, the Origin sends a connection request assigned zero DST SID toward the Target through the internet. If a gateway that is encountered in the process of the IP forwarding is a CIP gateway, it intercepts the forwarding and attempts to setup local connection with the Origin, sending back a connection approve to the SRC SID at the Origin. Simultaneously, the CIP gateway1 rewrites the IP header of the connection request, and waits for connection approve from a yet unknown CIP gateway. By accident, gateway2 is a pure IP gateway does not support CIP, so it forwards the datagram in the ordinary IP forwarding manner. Then the gateway3 intercepts the connection request, and a local connection between gateway1 and gateway3 is to be established. Next, the gateway3 and the target succeed in local connection establishment, eventually, the origin and the target are fully connected as a CIP connection.

- SYN Flag

  SYN means connection request.

Note that CIP does not know about other CIP gateways at all, except for the target CIP hosts, for we can never foresightedly know topologies of available internets.

Connection approve or refuse

The peer CIP host that received the connection request sees if it can acquire a new SID and enough memory to satisfy the required buffer space in itself. If can, it sends back an approve message consists of following descriptors:

- Source Stream ID
  Now that we have a new SID, we inform it of the peer.
- Destination Stream ID
  DST SID is peer's SRC SID, that is, "Source" and "Destination" are exchanged each other. Hearafter this stream is identified by a four-tuple of SRC SID, DST SID, Source IP address, and Destination IP address until the connection teardown.
- SEQ
  SEQ indicates the expected next sequence number in the cumulative positive acknowledgement policy.
- WIN
  WIN contains receiver's available buffer size. That is, this means receiver's window advertisement.
- ACK Flag
  ACK means a connection approve, that is, acknowledgement of the connection request.

Otherwise a refuse message is sent back including descriptors described below:

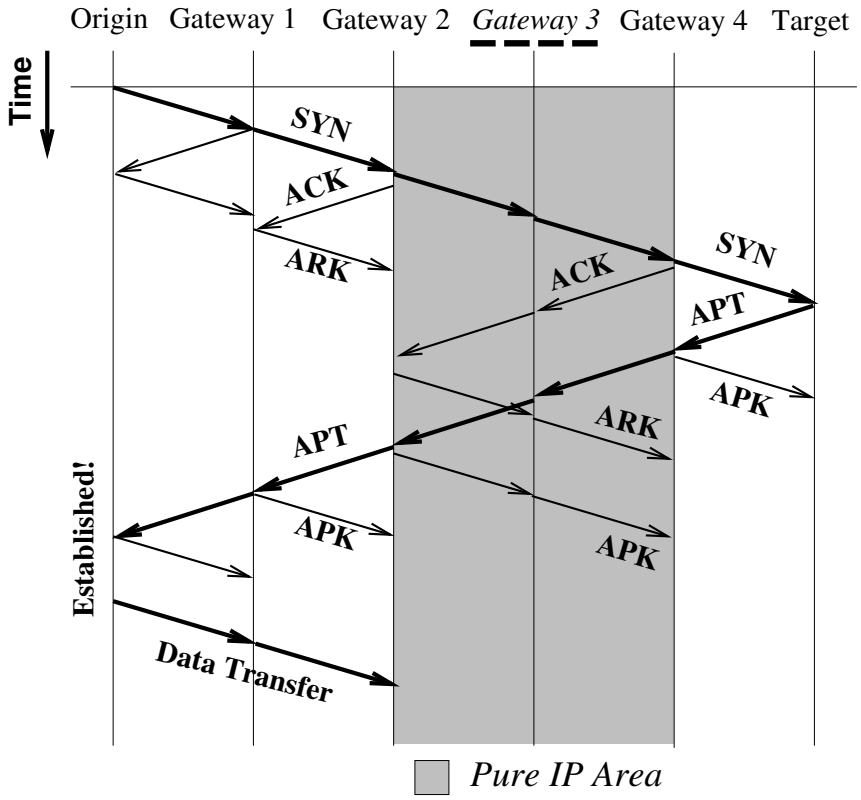
- SRC SID, DST SID
  Because the connection is yet to be established, SRC SID must be zero. DST SID is SRC SID of the peer which issued the connection request.
- ERR Flag
  ERR indicates a failure of connection setup, and validates the next CODE field.
- CODE
  CODE represents a reason of the error. Here, the host cannot setup CIP connection because all of SID allowed in it is exhausted, or it fails to acquire required buffer space.

If the CIP host is not the target, it propagates the connect message to the next-hop similarly as the first request.

In the mean while, the peer is wating this approve or refuse message. If this reply message is lost somewhere, the peer repeats the connect request again after some period. So we have to check SRC SID in the requests in order to avoid making duplicate SID to the same requests.

CIP agents received approve messages must acknowledge it in next format:

- SRC SID, DST SID
- ARK
  ARK means aknowledgement of the connection approve.



Here, Gateway3 is an original IP gateway that does not support CIP, so the section between Gateway2 and Gateway4 becomes a pure IP area. But Gateway2 and Gateway4 are able to communicate with each other, since CIP datagrams can be delivered as IP ones.

Figure. 3.5: CIP Connection Setup Diagram

The peer is waiting this ARK message. If it is lost somewhere, the peer sends ACK message again after some period. But in that case, a preceding connection accept message and corresponding handshake described afterwards also compensates for the lost ACK, ARK messages.

Connection accept

If the connection request reached the target, and it succeeded in aquiring required resource, the target sends back an accept message which contains following descriptors to its peer:

- SRC SID, DST SID
- SEQ
  SEQ indicates the expected next sequence number in the cumulative positive acknowledgement policy.

- WIN
  WIN contains receiver's available buffer size. That is, this means receiver's window advertisement.

- APT Flag
  APT means connection accept, which indicates that the target is ready to receive data.

Intervening gateways propagate back this message one after another. When the connection accept message is eventually propagated back to the origin, the connection is established fully, and the origin can use this connection for data transfer.

CIP agents received accept messages must acknowledge it in next format:

- SRC SID, DST SID

- APK
  APK means acknowledgement of connection accept.

The peer is waiting this APK message. If it is lost somewhere, the peer sends APT message again after some period. But in that case, a preceding data transfer also compensates for the lost APK message.

3.4.2 Connection teardown

It is allowed only for origins to teardown connections normally. Connection teardown message is composed as follows:

- SRC SID, DST SID

- FIN flag
  FIN means the end of the stream and connection teardown.

This message is carryed in the original sequence position, so it never skips the former datagrams. As soon as the message is acknowledged, a section of the whole connection is torndown partially. Meanwhile the message is propageted toward the target. State information of the closed connection persists while some period, ensuring the teardown.

3.4.3 Exceptional cases

Unexpected connection teardown

Connection teardown by intermediate nodes or targets is dealt with as an exceptional case. As yet, an exceptional reason code is only defined for this purpose.

- SRC SID, DST SID
- ERR Flag
  ERR indicates that an error happen, and validates the next CODE field.
- CODE
  CODE represents a reason of the error. Here, it indicates unexpected connection teardown.

As soon as the message is acknowledged, a section of the whole connection is torndown partially. Meanwhile the message is propageted toward the both side, the origin and the target.

Acknowledgement for the ERR message is composed as below:

- SRC SID, DST SID
- ERK Flag

Acknowledgement for the ERK message does not exist. State information of the closed connection persists while some period, ensuring the teardown.

3.5 Data Transfer

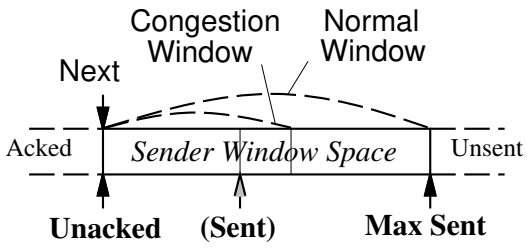
CIP data transfer is performed by credit-based hop-by-hop sliding window flow control like TCP end-to-end flow control. We indeed incorporated the existing technologies of fertile TCP to our protocol. This flow control also acts as a congestion detection mechanism, and furthermore performs congestion avoidance. Even though CIP takes the same measure to detect congestions as TCP, it is notable that intermediate gateways can detect immediate congestions and informs backward-hops of the fact through window advertisements directly.

3.5.1 Flow control

We use the credit-based hop-by-hop sliding window flow control as the principal congestion contorol mechanism. CIP's flow control resembles TCP's one. CIP has two windows:

- Normal window
  Receiver must advertise the current available buffer size, so sender can restrain the flow in response to it. In other words, this mechanism prevents receiver's buffer from overflowing.

  Now that intervening gateways advertise their own window size, when there is a congested network on a connection-path, window advertisement of the gateway at the front of the congestion behaves as a kind of a signal that directly tells backward-hops there is



Here, helvetica letters represent the reflected state of the receiver side, while **times-bold letters** represent the state in the sender's window space.

Figure. 3.6: CIP Sender Window

congested network beyond the gateway, which propagetes back and restrains the flow through the connection.

- Congestion window

  Sender has another window that we call a congestion window, which is independent of receiver's window advertisement and always takes precedence over it. Whenever datagrams are lost, sender reduces the congestion window as a symptom of a congestion and succeedingly increases it gradually, seeking an adequate point of the flow to ease the congestion.

  Now that intervening gateways can detect the immediate congestions, in that case, it is not necessary for an origin to depend upon its sole congestion window throughout the connection.

Receiver's window advertisement is delivered on the CIP header. Receiver also notifies the sender of the expected next sequence number. These are reflected in the sender's window space as depicted in Figure.3.6.

Sender manages three figures:

- Unacked

  The smallest unacknowledged sequence number, which always accords with receiver's requirement of the next sequence number.

- Sent

  Currently sending datagram's sequence number, which is usually equal to Max Sent except for the case of retransmission. Sent is always less than (Next + Congestion Window).

- Max Sent

  The maximum number that sender has ever sent. Max Sent is always less than (Next +

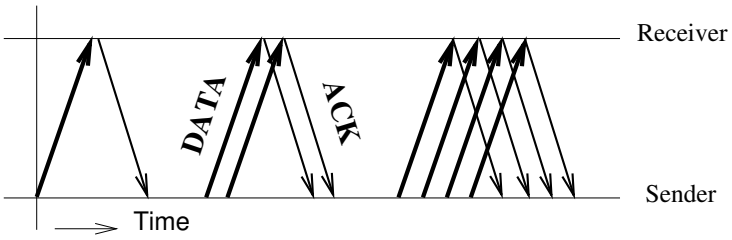


Figure. 3.7: Slow-start

Normal Window).

Slow-start

When starting a new connection or restarting it after a congestion, we set the congestion window to one packet size in order to preserve the internet we use from another congestion due to the new flow. Hereafter the congestion window grows one packet size every time the corresponding acknowledgement arrives to the sender, consequently the connection flow increases gradually. Although this starting mechanism is called as slow-start, it is not slow at all, for the congestion window grows exponentially every round trip time as shown in Figure.3.7. So the congestion window opens quickly enough to have a negligible effect on performance, while it makes sure that the congestion window size is within the current equilibrium point.

Congestion avoidance step

However, there is posibility that the exponential growth of the congestion window accounts for another congestion when restarting the stream after one congestion. So we keep the size of half a unacked window as a threshold of restraint, then we control the growth of the congestion window as follows:

- If the current congestion window size is less than the threshold, we leave the growth as it is, that is, it goes on growing exponentially.
- If the current congestion window size exceeds the threshold, hereafter we restrain the growth linearly as shown in Figure.3.8.

3.5.2 Error control

CIP estimates smoothed round-trip time described below. Whenever one datagrams's round-trip time exceeds the estimated one, CIP infers that it was lost and retransmit it. We call such error corrective control which ensures reliability of streams as error control.

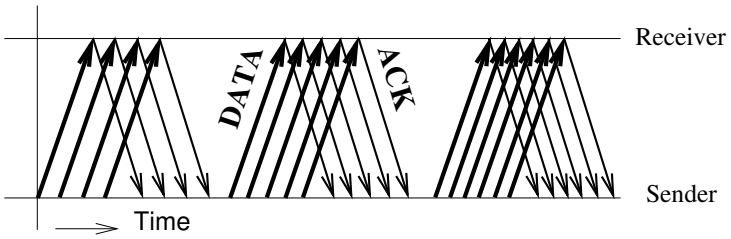


Figure. 3.8: Congestion Avoidance Step

Estimation of smoothed round-trip time

We measure round-trip time about each non-retransmitted datagram and calculate their mean and variation[6] to estimate smoothed round-trip time and determine the retransmission time out. Our algorithm is derived from Jacobson's one which is described clearly in [Jac88], so we explain it briefly here.

Smoothed round-trip time is typically represented as following:

$$A \leftarrow (1-g)A + gM$$

where A is the average round-trip time, M is a new measured sample of round-trip time, and g is a "gain" $(0 < g < 1)$. We rearrange and collect terms multiplied by g:

$$A \leftarrow A + g(M - A)$$

We also calculate the mean deviation, D, which has a relationship that:

$$D \simeq \sqrt{\frac{\pi}{2}}\sigma$$

where σ is the standard deviation of SRTT ($\sigma^2 = Variation$). Here we estimate the final retransmission time out as below:

$$Err \equiv M - A$$

$$A \leftarrow A + gErr$$

$$D \leftarrow D + g(|Err| - D)$$

$$RTO = A + 2D$$

Herein the queuing theory says that [Com91]:

[6] In fact, we calculate their mean deviation ($|Measure - Average|$) which has good correlation with their variation.

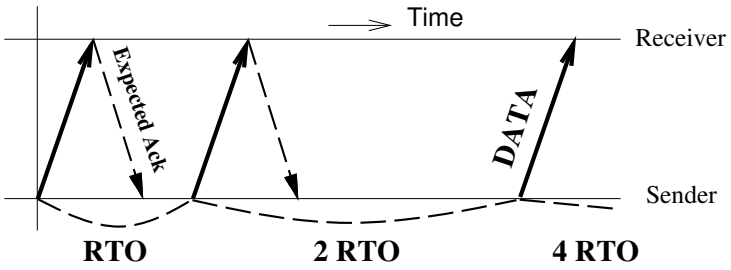


Figure. 3.9: Retransmit Timer Back-off

$$\sigma \propto \frac{1}{1-L}$$

where L is a network load ($0 \leq L \leq 1$). Because our estimation contains this term as a whole, it is adaptive to diverse variations encounterd in internets.

Retransmission

When a retransmission time out occurs, we then does not renew smoothed round-trip time, instead we apply the exponential retransmit timer back-off to succeeding retransmitted packets, that is, the retransmit timer is multiplied exponentially every packet loss (as shown in Figure.3.9.). These policies for retransmission described above is known as "Karn's algorithm"[KP87]. Meanwhile, we set the congestion window to one packet size for the sake of slow-start.

3.5.3 Rate control

We attempt to apply rate control to the connection-paths through which streams of packets are delivered, including pre-allocation of buffers. However, we cannot guarantee any amounts of transfer rate by CIP rate control, since we do not yet reserve any network bandwidth and/or packet switch processing bandwidth at all. Instead, CIP rate control only provides with a method to regulate upper bound of transfer rate of streams at endpoints and each intermediate system. Note that actual transfer rate can be less than pre-specified one.

Parameter

CIP rate control is performed just like XTP's one described in Section. 2.2.2. CIP manages two parameters as follows:

- Rate parameter
  The rate value specifies in bytes per second the maximum rate at which data can be

consumed by the receiver.

- Burst parameter
  The burst specifies the maximum number of bytes which can be consumed in one burst of packets, that is, packets sent in rapid succession.

The quotient *burst*/*rate* gives us the period of time during which *burst* number of bytes can be transmitted. A sender always collects the total length of acknowledged packets during the period, and verifies that the total size does not exceed *burst* value. As soon as it detects an overrun of the transfer rate, it suspends transmission control and waits until the end of the burst period. After that, the sender resets the total length to zero, and repeats above process.

Chapter 4

IMPLEMENTATION

We implemented the new protocol CIP, on Mach 3.0[Loe92]. Although the current CIP software is yet only a prototype, it includes almost all specifications described in the previous chapter.

Mach has following advantages:

- Network protocol decomposition from the kernel implementation.
  Mach Micro Kernel provides us with a development environment suitable for protocol implementations, for there is no need to struggle with monolithic kernels such as UNIX.

- Existence of real-time primitives.
  (RT-)Mach[HTNH91] provides with real-time primitives, for example, high resolution timer, real-time interprocess communications, real-time threads that are executed according as their precedence in various scheduling policies prepared in advance.

So far we do not utilize such real-time primitives effectively. But we think the second point will be of great advantage in the future for handling real-time network flows.

Hereafter we describe our implementation in detail.

4.1 Implementation Overview

In the first place, we explain our implementation overview briefly. CIP the network layer protocol was implemented as a Mach task, including other components of the protocol stack. for example, Ethernet handling, Internet Protocol, and transport protocols. CIP software is written in the language C, and is mainly composed from following primitives:

- Function
  Ordinary procedures represented in the form of C functions, which are invoked by other functions as a method of synchronous communication.

- Thread
  Mach supports the multi-task, multi-thread environment. The substance of threads are ordinary functions. In our implementation, threads activated by CIP software are executed in the manner of time sharing based on their priorities.

- Buffer
  The most general data unit in our implementation. Buffers are used to store all of data to be made for datagram construction and delivery throughout CIP software.

- Queue
  For sake of asynchronous communications among threads, we prepared a FIFO queuing mechanism of Buffers.

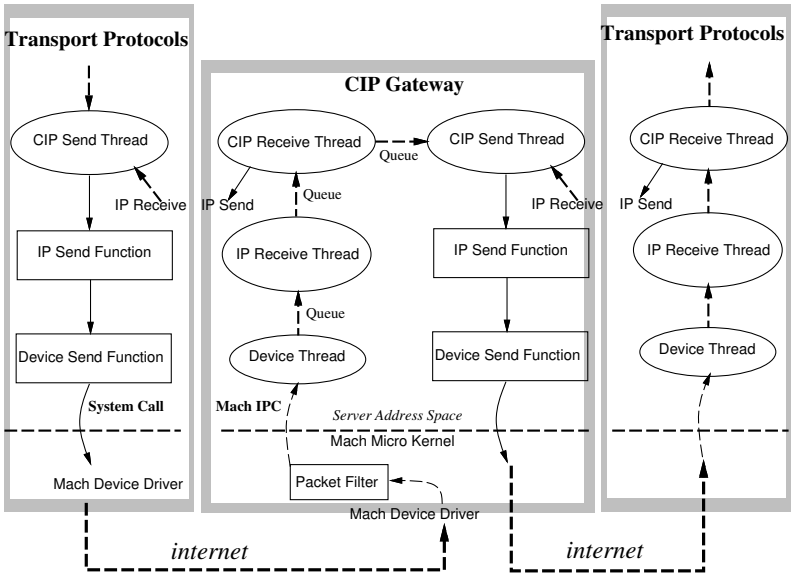


Figure. 4.1: Software Structure

Using above primitives, CIP software performs internetworking procedures. In particular, threads play important role as networking entities. CIP software structure is shown in Figure. 4.1. The detail of them are described afterwards.

4.2 Fast Memory Management

As concerns networking that requires time critical performance, memory management is a key part of its implementation. Some operating systems such as UNIX[SMMJ89] and network protocol supporting software such as x -kernel[HP91] prepare equipments for fast memory management. We provide similar memory management equipments which enable us to allocate some amounts of memory quickly or deallocate them fast.

4.2.1 Buffer

Protocol implementations usually need push/pop operations for a pair of header and data portion in a buffer (as shown in Figure. 4.2). We support such operations by an abstract data type "Buffer" which enables implementors to manage buffers like *mbuf* of UNIX. In Figure. 4.2, bold-line depicts regions stored in Buffer.

Prepared operations are following:

- Constructor/Deconstructor

```
buffer_t *BufferAlloc( int size );
```

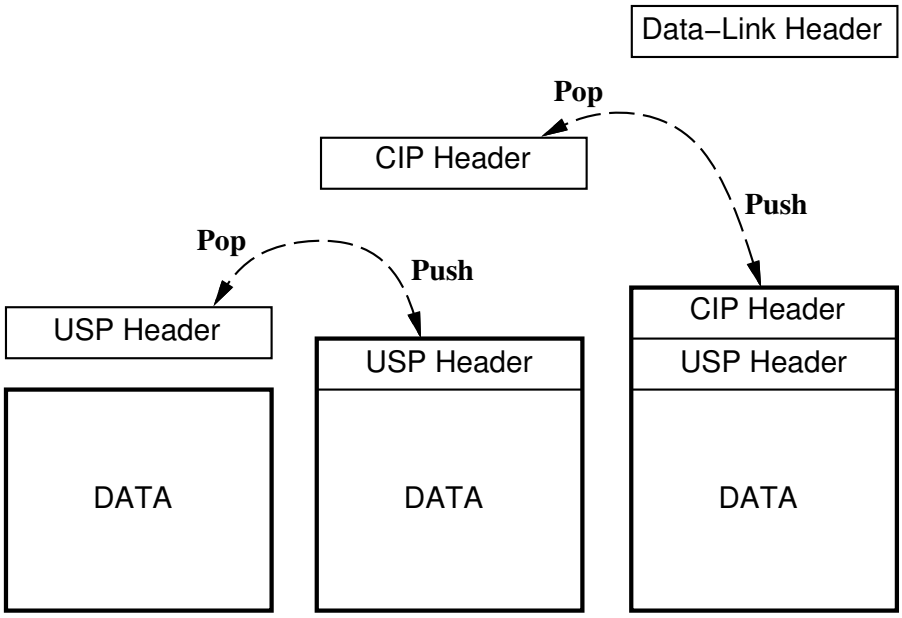


Figure. 4.2: Buffer Operation

```
boolean_t BufferDealloc( buffer_t *buf );
```

`BufferAlloc` takes an integer value which indicates the length of a new buffer, and returns the buffer. `BufferDealloc` deconstructs the buffer.

- Push/Pop operation

```
char *BufferPush( buffer_t *buf, int len );
boolean_t BufferPushCopy( buffer_t *buf, int len, char *dat );
char *BufferPop( buffer_t *buf, int len );
```

`BufferPush` takes an integer value as its argument and extends the data region of the buffer, and returns a pointer to the extended data region. `BufferPushCopy` takes an additional argument which indicates existing data to be copied into the region. `BufferPop` acts contrary to `BufferPush`.

- Pseudo-copying

```
buffer_t *BufferCopy( buffer_t *src );
```

`BufferCopy` takes a buffer to be copied, and returns the copied buffer. This is only pseudo-copying, for the substance of the source buffer is not copied.

- Modifier

```
char *BufferTop( buffer_t *buf );
boolean_t BufferOverride( buffer_t *buf, int size );
boolean_t BufferTrunc( buffer_t *buf, int size );

int BufferLength( buffer_t *buf );
char *BufferHead( buffer_t *buf );
```

`BufferTop` returns an internal pointer to the buffer top. `BufferOverride` shrinks the buffer size into `size` from the buffer top. `BufferTrunc` truncates the data length stored in the buffer.

`BufferLength` returns the length of stored data. `BufferHead` returns a pointer to the data top.

4.3 Event Management

Mach provides with mutual exclusion equipments, for example, mutex lock, unlock, and condition wait, signal. We abstract them as an abstract data type "Mutex", and use it as a tool for event management.

- Constructor/Deconstructor

```
rt_mutex_t *MutexAlloc();
boolean_t MutexDealloc( rt_mutex_t *mx );
```

`MutexAlloc` constructs a Mutex. `MutexDealloc` deconstructs it.

4.3.1 Timer

Protocol implementations usually need timer operations for measuring various statistics. Mach provides with high resolution timer which implementors can access via particular locations.We prepare an external valiable `mappedClock`:

```
extern timespec_t *mappedClock;
```

which represents the local system time in the form of `timespec_t`. Type `timespec_t` is a Mach primitive and contains two data fields of `seconds` and `nanoseconds`.

We also provides an abstract data type "Event" which issues timer events by using Mutex.

- Constructor/Deconstructor

```
event_t *EventAlloc( rt_mutex_t *mx, timespec_t timeout );
boolean_t EventDealloc( event_t *ev );
boolean_t EventCancel( event_t *ev );
```

`EventAlloc` allocates a timer event, which raises a condition signal at `timeout`. `EventCancel` cancels an allocated event and returns the final status of the event, that is, FALSE in the case that it was already raised, otherwise TRUE. `EventDealloc` is light weight version of `EventCancel`.

4.3.2 Queue

We also abstract a data type "Queue", which provides with FIFO Buffer queues. Queue prepares two types of signals, that is, internal signals and external signals. Internal signals are hidden from implementors and used for following purposes:

- Inqueue with blocking.
  When we try to insert a buffer into a queue with blocking, if the queue is filled up, the continuation releases CPU and waits on the condition which is prepared by the queue. When others dequeue a buffer from the queue, an internal signal is issued and the waiting continuation is activated.

- Dequeue with blocking.
  When we try to dequeue a buffer from a queue with blocking, if the queue is empty, the continuation releases CPU and waits on the condition which is prepared by the queue. When others inqueue a buffer into the queue, an internal signal is issued and the waiting continuation is activated.

External signals are prepared by implementors and used for following purposes:

- Notify inqueuing.
  Whenever someone inqueues the queue, an external signal is issued and notify the fact of the implementor set the signal.

- Notify dequeueing.
  Whenever someone dequeues the queue, an external signal is issued and notify the fact of the implementor set the signal.

Prepared operations are following:

- Constructor/Deconstructor

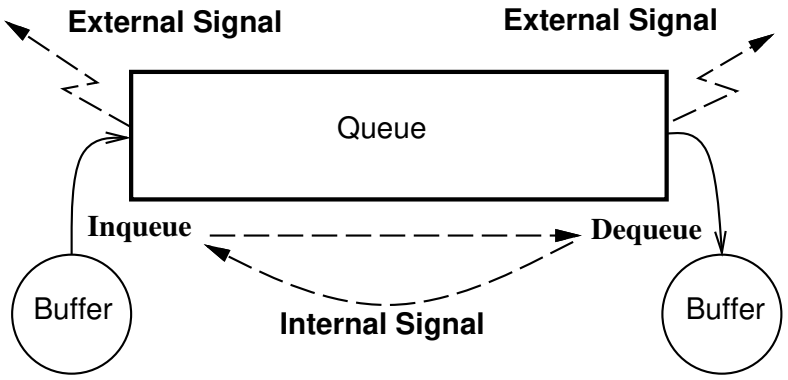


Figure. 4.3: Queue Operation

```
queue_t *QueueAlloc( int size, rt_mutex_t *mxI, rt_mutex_t *mxD );
boolean_t QueueDealloc( queue_t *que );
boolean_t QueueSetMutex
                ( queue_t *que, rt_mutex_t *mxI, rt_mutex_t *mxD );
```

`QueueAlloc` allocates a queue, where `mxI` and `mxD` are external mutual exclusions to be set on the queue. `mxI` is used to notify inqueuing, while `mxD` is used for dequeuing. Here NULL pointer affects nothing. `QueueSetMutex` allows mutexes to be reset.

- Modifier

```
int QueueLength( queue_t *que );
boolean_t QueueStandby( queue_t *que );

boolean_t QueueInsert( queue_t *que, buffer_t *dat );
boolean_t QueueInsertNB( queue_t *que, buffer_t *dat );
buffer_t *QueueDelete( queue_t *que );
```

`QueueLength` returns the total length of stored buffers. `QueueStandby` checks whether there are some buffers in the queue or not.

`QueueInsert` performs inqueue with blocking. `NB` means no blocking. If there is no buffer in the case of inqueue with no blocking, FALSE is returned. `QueueDelete` performs dequeue with blocking.

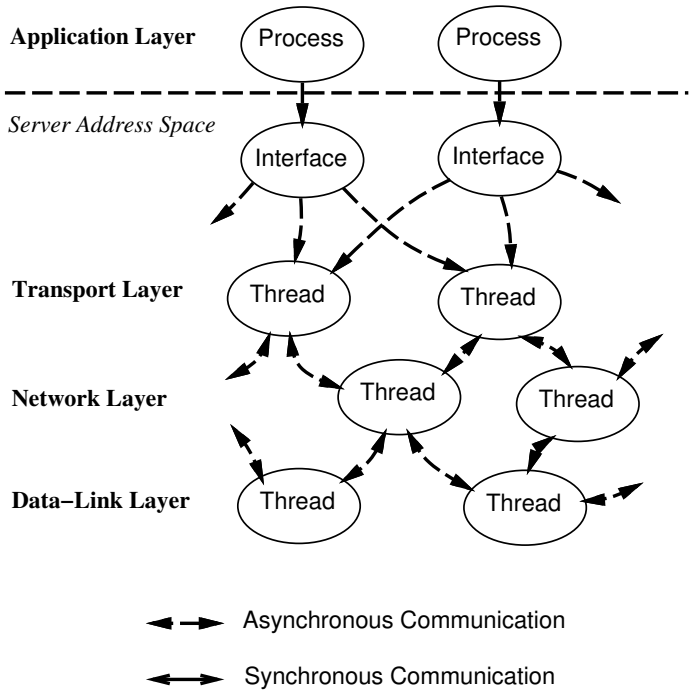


Figure. 4.4: Thread Layout Policy

4.4 Thread Layout Policy

4.4.1 Thread

We provide an abstract data type "Thread" as follows:

- Constructor/Deconstructor

```
rt_thread_t *ThreadAlloc( void (*func)(), int *arg, int pri );
boolean_t ThreadDealloc( rt_thread_t *th );
```

`ThreadAlloc` allocates a thread whose substance is `func` which has an argument `arg`, and activate it at once. `pri` means precedence of the thread. `ThreadDealloc` deallocates the thread.

4.4.2 Layout

Our networking software is organized from aggregate threads based upon asynchronous communication. Whenever a rendezvous is required, it is done as a condition waitng of mutual exclusions. Indeed, there is no busy-waiting loop in our implementation. We layout a group of threads for each layer of protocol stack as shown Figure. 4.4.

4.5 Device Input Output

Device (data-link interface) input is performed as Mach IPC (interprocess communication) from Mach Micro Kernel to a device thread. Mach examines incoming packets according to registered packet filters, and dispatches them to particular IPC ports. Device output is performed by invoking functions.

For the sake of modularity, we keep device modules and CIP modules from depending upon each other. That is, they are implemented individually except for interfaces.

Device handling has two step abstraction as below:

- Device Open as Generic Function
  Implementors open devices by `DeviceOpen`.

  ```
  typedef void (*receive_thread_t)( dev_t * );
  typedef boolean_t (*filter_func_t)( filter_t *, int * );
  dev_t *DeviceOpen
         ( char *name, int priority, filter_func_t f, queue_t *que );
  ```

  where `name` indicates the data-link interface name, `f` means a filter-construction procedure, `priority` means the precedence of the packet filter, `que` indicates a queue used for asynchronous communications between IP receive threads and device threads described afterward.[1]

- Device Proper Procedures
  We prepare device proper procedures for each device. For example, we provide following functions for Ethernet.

  ```
  boolean_t EtherFilterFunc( filter_t *filter, int *size );
  void EtherSendFunc( dev_t *dev, buffer_t *buf, char *peer );
  void EtherReceiveThread( dev_t *dev );
  ```

  `EtherFilterFunc` is used by `DeviceOpen`. Implementors must allocate device threads whose substance is `EtherReceiveThread` for every device. `EtherSendFunc` is used for output of Ether packets by implementors.

Once device is opend and device thread is activated, the device thread communicate with other threads asynchronously, using Queue which carries data received at the device and stripped off data-link headers by each device proper procedures.

[1] `DeviceOpen` does not allocate any device thread.

4.6 CIP Interface

Because CIP is a connection-oriented internetworking protocol, upper protocols must get a service access point for every stream before data transfer. There are two types of service access points (SAPs) as follows:

- SAP for active open

  ```
  sid_t *CipConnect( ip_addr_t targetIp, buffer_t *buf,
                     unsigned short window,
                     unsigned short rate, unsigned short burst,
                     unsigned char protocol );
  ```

  where `targetIp` is a target IP address, `protocol` is upper protocol's protocol ID. We allow upper protocols to communicate with their peer, including `buf` to the first CIP connect request message. For the experimental purpose, `window`, `rate` and `burst` are allowed to be specified. `window` indicates a required sliding window size, `rate` means an upper bound of transfer rate, `burst` is used for rate control. `CipConnect` returns a SAP for active open.

- SAP for passive open
  Every upper protocol must provide with a procedure for CIP to notify connection setup to upper protocols like below:

  ```
  boolean_t UspCipNotifyConnected( sid_t *sid, buffer_t *buf );
  ```

  where `sid` means a SAP for passive open, and `buf` is a message from peer of the upper protocol. CIP choose these notify function according as a protocol field of IP header, and invoke them whenever a connection is established.

Hereafter the CIP connection is distinguished by the SAP.
Connection teardown is performed by following functions:

- Close active opened connection

  ```
  boolean_t CipClose( sid_t *sid );
  ```

- Abort passive opened connection

  ```
  boolean_t CipDisconnect( sid_t *sid );
  ```

Meanwhile, CIP (data) input output is performed through the SAP data structure. Herein, these SAPs are revealed only to transport protocols. Upper protocols must not reveal them to users.

4.7 CIP Input Output

CIP input output is apparently asynchronous. It is performed through queues which belong to particular SAPs. That is, upper protocols insert buffers into the queue, or dequeue buffers from the queue. But internal mechanisms are quite complex. We explain them in detail here.

4.7.1 Further abstraction of device abstraction

CIP further abstracts device abstraction as IP device, which is constructed as follows:

```
ip_dev_t *OpenIpDevice( char *name,
                        ip_addr_t ip, ip_addr_t mask,
                        filter_func_t f,
                        receive_thread_t r, send_func_t s );
```

where `name` is a device name that is passed to `DeviceOpen`, `f` is filter function which is passed to `DeviceOpen`, `r` is a thread which receives datagrams from Mach Micro Kernel, that is, this is the device thread, `s` is a function by which one can send packets to the device, `ip` is the device's IP address, `mask` is a subnet mask.

We can recognize "IP device" as an abstraction of "IP address and data-link device", and realize a strong abstraction barrier between data-link layer that belongs to data-link address arena and CIP (or IP) layer that belongs to IP address arena.

4.7.2 IP input

CIP allocates an IP receive thread for each device thread. An IP receive thread receives device input through the device queue allocated precedingly, and dispatch them to transport protocols (as shown in Figure. 4.5.)

However, if there are CIP datagrams in device input, IP receive thread behaves differently from ordinal IP functions.

- If it is a connection request,
  IP receive thread allocates a new SID and,
  - If the host received the request is a target,
    IP receive thread notifies the establishment to the transport protocol which corresponds to the protocol field of IP header.
  - Otherwise,
    IP receive thread allocates another SID, and propagetes the connection request to a next-hop.

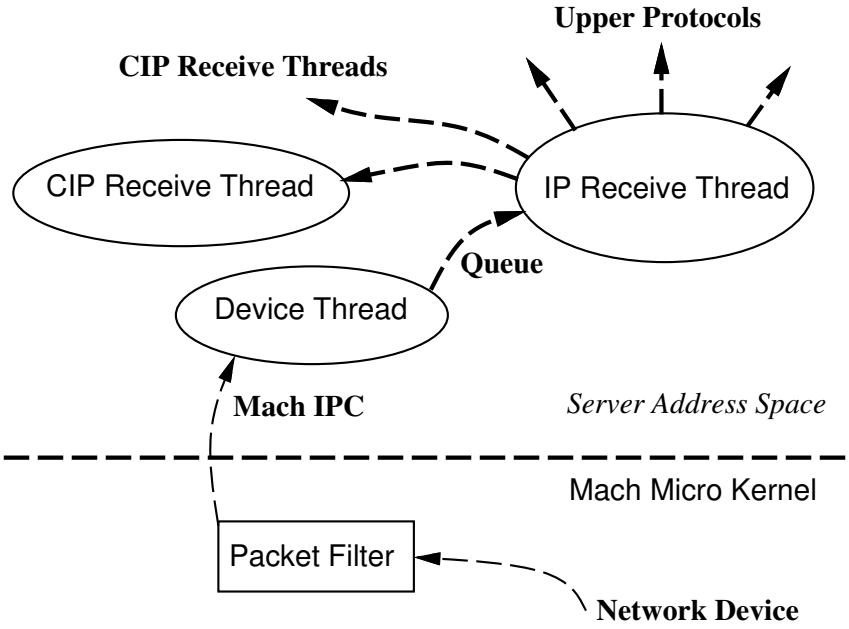


Figure. 4.5: IP Receive Thread

- If it is a message of an established connection,
  - If the host received the request is a target,
    IP receive thread passes it to a particular CIP receive thread directly.
  - Otherwise,
    IP receive thread forwards it as an IP datagram.
- Otherwise,
  IP receive thread discards it.

4.7.3 IP output

IP output is performed by invoking following function:

```
void IpSend( buffer_t *buf, int optLen, route_t *route,
             ip_addr_t srcIp, ip_addr_t dstIp,
             unsigned char protocol );
```

where `buf` is a buffer to be sent, `optLen` is length of the IP options stored in the datagram, that is, IP options are already pushed, `route` is an optional routing information, `protocol` indicates which protocol this datagram belongs to.

4.7.4 Routing and address resolution

OSPF routing function is yet to be implemented. Instead, we now use a fixed routing table for our experimental internet.

Given the target IP address, the routing function finds next-hop's IP address and return an IP device to reach the next-hop.

```
ip_dev_t *Route( ip_addr_t ipAddr, unsigned char tos, ip_addr_t *nextHop );
```

where `ipAddr` indicates a target IP address, `tos` corresponds to the TOS field of IP header which means the type of service required for the datagram, `nextHop` is to be assigned next-hop's IP address, and `Route` returns an IP device.

And given next-hop's IP address, the address resolution function returns its data-link address, for example, Ethernet address.

```
char *Resolve( ip_dev_t *ipDev, ip_addr_t nextHop );
```

where `ipDev` indicates the IP device which `Route` returned, `nextHop` indicates the next-hop. Note that the next-hop is on the same sub-network as the host issuing this function does. Herein, interpretation of the result depends on the sending function of the IP device which `Route` must return.

4.7.5 CIP Thread

There are two types of CIP threads, one is CIP send thread, another is CIP receive thread. `CipConnect` activates a CIP send thread in the face of active open, while CIP receive thread is allocated by an IP receive thread in the face of passive open. CIP thread layout is shown in Figure. 4.6.

First of all, CIP send thread must setup a connection. Connection setup is performed in a manner described in Section. 3.4.1.

CIP send thread always watches its SAP's queue, and if there are some buffers, it dequeues them and sends them in a manner described in Section. 3.5.[2] Meanwhile, CIP receive thread always watches an internal queue into which IP receive threads insert buffers. If there are some buffers in the queue, it receives them and eventually passes them to transport protocols.

If there is necessity to relay data and control messages, a pair of CIP send thread and CIP receive thread works as a gateway (as shown in Figure. 4.7.). The queue between both threads originally belongs to send thread side, but it is shared among them.

[2]Since CIP threads works honestly in the manner described in Section. 3.5, we dare not explain about CIP threads coding closely. We describe only the smoothed round-trip time estimation method in Appendix.

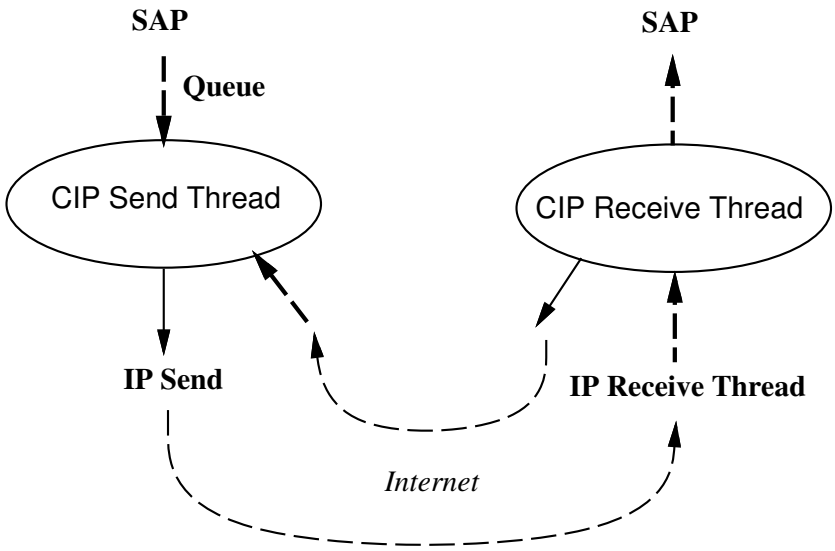


Figure. 4.6: CIP Thread Layout

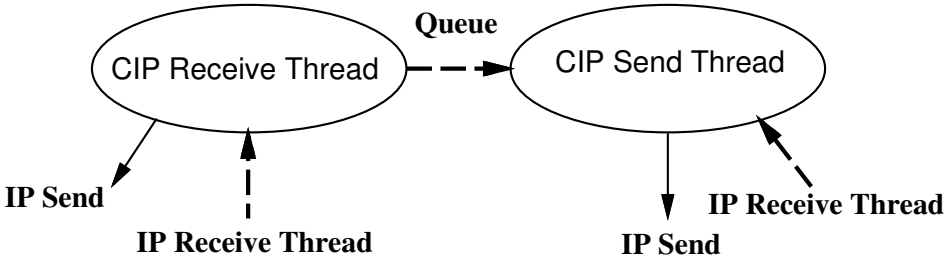


Figure. 4.7: CIP Gateway

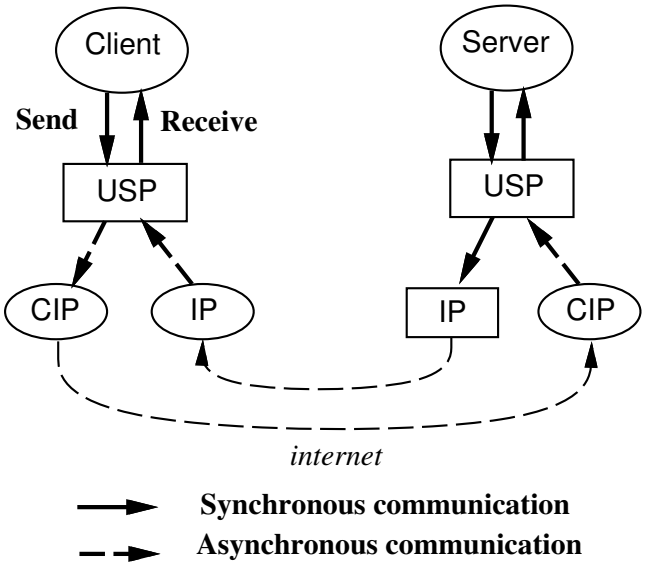


Figure. 4.8: USP Connection

4.8 Transport Protocol Input Output

We implemented a new experimental transport protocol, USP (User Stream Protocol), as an example of a transport protocol implementation. We can also implement transport protocols such as TCP, UDP, on our framework, but these implementation are yet to be done.

4.8.1 User Stream Protocol

USP (User Stream Protocol) is a new transport protocol. As UDP (User Datagram Protocol) provides us with simple abstraction of IP, so USP offers abstraction of CIP similarly. USP reveals USP ports like UDP or TCP ports to users, and provides users with USP streams as reliable as CIP streams. Even though CIP constructs one-direction connections, USP offers duplex connections, where the direction from an origin to a target is setup using CIP, while the reverse direction from a target to a origin is setup using IP (as shown in Figure. 4.8.)

USP interfaces are following:

- Active open

```
usp_t *UspConnect( ip_addr_t peerIp, unsigned short peerPort,
                   unsigned short window,
                   unsigned short rate, unsigned short burst );
```

where `peerIp` is a peer IP address, `peerPort` is a target server's USP port. For the experimental purpose, `window`, `rate` and `burst` are allowed to be specified. `window`

indicates a required sliding window size, `rate` means an upper bound of transfer rate, `burst` is used for rate control. The client's source USP port is choosen arbitrary, avoiding duplication within the same host. An origin IP address is determined after routing to a target. Returned pointer typed "`usp_t *`" works as a SAP in USP.[3]

- Passive open

```
usp_t *UspListen( unsigned short myPort );
boolean_t UspAccept( usp_t *usp );
```

where `myPort` is a passive opened server USP port. Servers invoke `UspListen` and obtain a USP data structure, abstraction of a USP stream. Then servers call `UspAccept` and wait connection notifies there. An IP address of a server is determined during a connection setup.

- Data transfer

```
boolean_t UspSend( usp_t *usp, char *str, int length );
boolean_t UspReceive( usp_t *usp, char *str, int *length );
boolean_t UspReceiveStandby( usp_t *usp );
```

where `str` is a pointer to a character array space, `int length` indicates the actual length of the character array to be sent, `int *length` returns the length of data received actually. `UspReceiveStandby` checks whether there are buffers in the receive queue or not.

- Close

```
boolean_t UspClose( usp_t *usp );
```

`UspClose` requests CIP to teardown the CIP connection used in the USP conversation. Differences between `CipClose` and `CipDisconnect` are absorbed in the `UspClose` function.

4.8.2 Multiplex and demultiplex

USP has a following header:

- Source port (2 octets), Destination port (2 octets)
  Source port and Destination port contains an user specific number that identifies an USP stream.

[3] In general, these SAPs should not be revealed to users in such a easy style, but we leave it as it is, for we do not implement USP skillfully yet.

- Data length (2 octets)
- Data checksum (2 octets)
  Not implemented. Data checksum should be used to ensure end-to-end reliability.

Using above USP header format, USP multiplexes USP datagrams into IP datagrams. In the mean time, USP does not need to multiplex them into CIP datagrams, since USP passes those required information in the first connection request, and succeedingly only uses a stream which was created for the connection.

IP datagrams which have the particular protocol id defined for USP found in the protocol field of IP header are demultiplexed to the USP "protocol handler", whose substance is a function takes an argument, buffer. And the USP protocol handler demultiplex these USP datagrams to each USP port.

In the case that we implement another transport protocols like TCP or UDP, we also prepare a "stream handler", whose substance is a thread. The reason we need to prepare such handlers is stems from the fact found in our thread layout, in particular, queue arrangement among those threads, for the queue between a CIP receive thread and a stream handler belongs to not the handler but the CIP thread. Stream handlers dispatch or concentrate some CIP streams to ports belong to each protocol. Figure. 4.9 and 4.10 depicts the state of affairs surrounding multiplexing and demultiplexing, respectively.

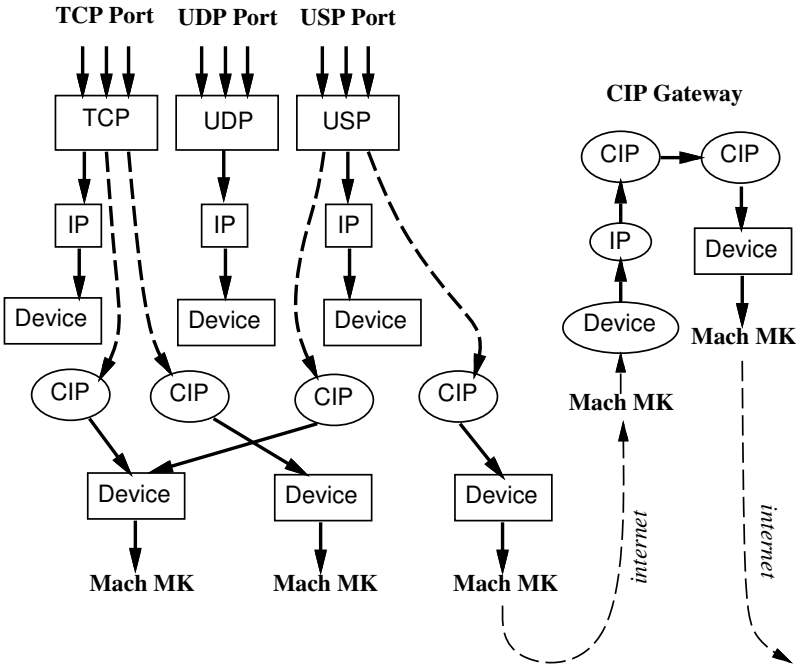


Figure. 4.9: CIP Multiplexing

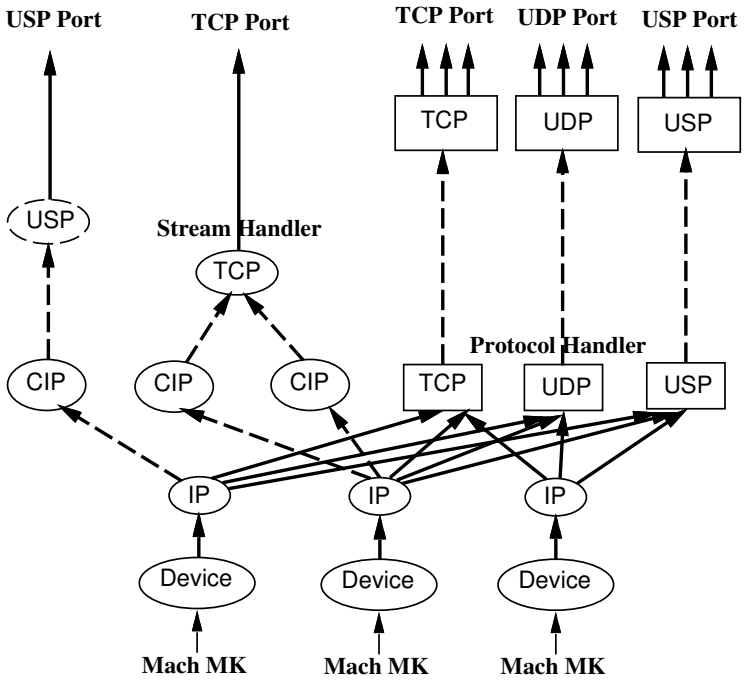


Figure. 4.10: CIP Demultiplexing

Chapter 5

EVALUATION

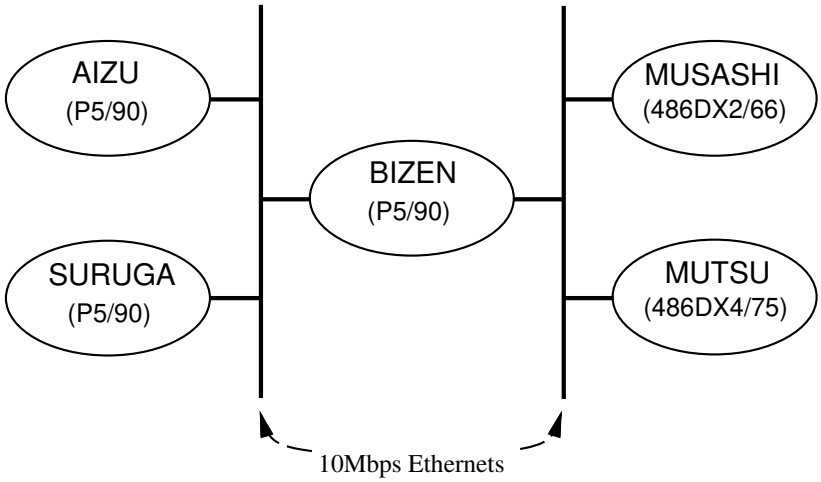


Figure. 5.1: Our Experimental Internet

Here we evaluate our implementation, examining its performance in the face of several aspects, in particular, including comparison between CIP architecture and IP one, and discuss usability of our framework.

We measured all statistics on the experimental internet depicted in Figure. 5.1, where AIZU (Pentium 90MHz, 3Com509 10BaseT Ethernet card), BIZEN (P5/90, 503-T), SURUGA (P5/90, 509-T), are connected to the main sub-network of our laboratory, while BIZEN (P5/90, 3Com509 10BaseT), MUSASHI (486DX2 66MHz, 509 10Base5), and MUTSU (486DX4 75MHz, 589 10BaseT) are attached to the second one (Both networks are 10Mbps Ethernet). Since BIZEN have two device interfaces each of which reaches the respective network, it can work as a gateway that performs either a CIP gateway or a pure IP one.

The measurements were also done on following conditions:

- USP conversations as test data transfers.
  All of test data transfers were represented as USP conversations, where 1KB data size user messages were delivered from an origin to a target continuously.

- Collecting all timing data at an origin.
  Since it is hard to collect distributed timing data at several points synchronously, we measured all timing data at an origin, which determined the starting and ending time of each stream. Leaving times of each datagram at a CIP send thread in the standardlized time were stored temporarily in memory, and written down to a hard disk at the end of the measurement.

- Program interface in the form of a library.

We intend to provide with abstractions of transport protocols in the form of libraries, especially, send and receive operations should be offered in such style[MN93]. So the measured timing data include no cost spent for communications with servers.

- Execution in the ordinary working environment.
  Since we have not eptimized our implementation yet, we do not measure any break-down in detail. So we execute test data transfers in the ordinary working environment on Mach 3.0.

We prepared a pair of USP client and server programs which were used to execute USP conversation. Both programs are quite simple. USP client goes on sending 1KB size data to USP server, while USP server goes on waiting and receiving the data, and only discards them with doing nothing.

We activated two clients at the origin host. One client started at the origin time "zero", and another did one second after from the origin time. Both the former and the latter clients toredown their connection two seconds after from their starting time respectively. Meanwhile, we activated two servers at the target host(s). They did not always reside on the same host. Servers waited for connection requests from respective origin hosts, and went on receiving a series of test data after their passive open.

5.1 Simple Performance

We begin with examining simple performance of our implementation. Here simple performance means CIP performance measured through some USP conversations between a pair of a client and a server which were located within a sub-network where no other affective conversations were done, excluding interconnecting effects. Two USP conversations were done from MUSASHI to BIZEN, where sliding window size of their connections were set to 4KB. One conversation started, that is, the first connection request was issued at the origin time, and another conversation started one second after from the origin time.

We illustrate the results of conversations as shown in Figure. 5.2, where horizontal direction depicts the current time (sec), while vertical direction depicts the delivered sequence number, which can be converted into delivered user data size (KBytes), including neither USP header nor CIP header size.

The short delay of starting point of each line is due to the cost spent for connection setup, which tells it took about 0.082sec and 0.094sec to setup each connection. Inclination of each line represents corresponding transfer rate. We can see three areas divided in the figure as below:

- The first area only belongs to the former conversation from 0.0(sec) to 1.0(sec).

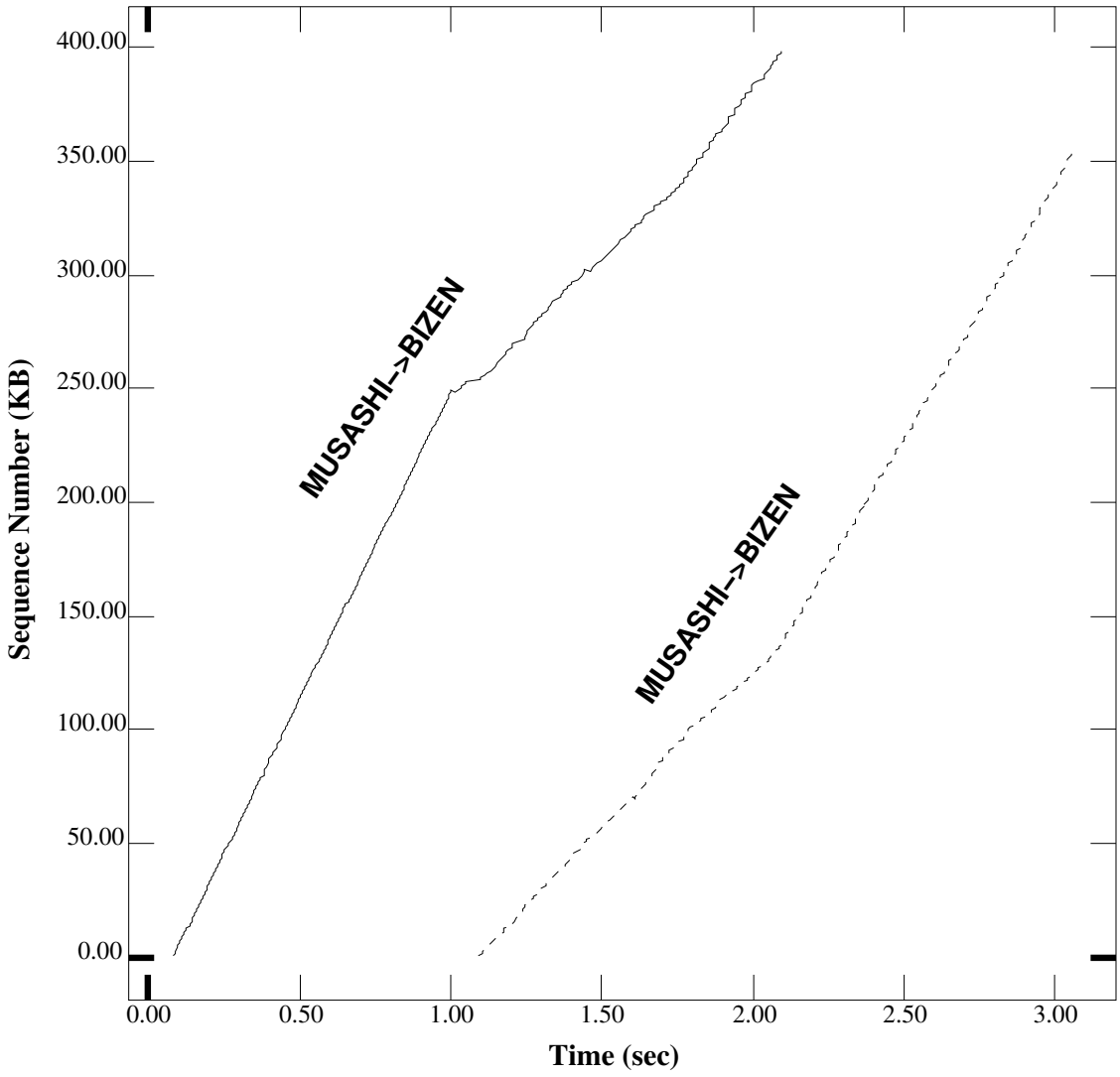


Figure. 5.2: Simple Performance

Simple Performance	0.0∼1.0	1.0∼2.0	2.0∼3.0[sec]
The former	269	132	
The latter		136	222
Total [KBps]	269	268	222

- The second area belongs to the both conversations from 1.0(sec) to about 2.0(sec). There are two conversations simultaneously.
- The third area only belongs to the latter conversation from 2.0(sec) to 3.0(sec).

In the first area, a pair of CIP entities, an origin and a target, obtained 269 KBytes/sec[1] of transfer rate, which means the maximun amounts of transfer rate allowed within their packet switch processing power or buffer space capability, since 269KBps (i.e., 2.1Mbps) does not reach the saturation point of 10Mbps Ethernet.

In the second area, one conversation obtained 132KBps, while another did 136KBps. Since 132KBps and 136KBps are almost the same and the sum of them (268KBps) approximately equals the first area's throughput (269KBps), we can say that the two conversations bisected the originally allowed transfer rate equally. Yet what made them so? As a matter of course, saturation of network bandwidth is not the answer as we pointed out above. Thinkable answers are following:

- Saturation of packet switch processing bandwidth
- Saturation of capability of buffer space

We estimated capability of buffer space by measuring one USP conversation throughput while changing its sliding window size from 1KB to 16KB (as shown in Figure. 5.3: all conversations were done from MUSASHI to BIZEN.)

According to the figure, the transfer rate in the first area was subject to sliding window size. In the second area, however, there is no constraint of buffer space, since 132KBps is far from estimated throughput (from 200Kbps to 300KBps). So we can infer that apparent equality of the first rate (269KBps) and the sume of the second rates (268KBps) is a great coincidence, and the reduction of each stream's throughput may be due to time sharing effects of Mach threads. To ensure that, we measured two USP conversations which were relatively unrelated to each other. One conversation was done from BIZEN to MUTSU, while another was done from BIZEN to SURUGA. Since neither of the hosts besides on the same sub-network, we can say that time sharing effects at BIZEN account for diminution of both throughputs found in the figure. The result is shown in Figure. 5.4.

In the third area, the sole conversation obtained the full allowed bandwidth again. The transfer rate was about 222KBps. The diminished throughput (from 269KBps to 222KBps) is maybe due to time sharing effects at MUSASHI and/or BIZEN, but we cannot assert further things beyond the fact that there is no constraint of network bandwidth and proved constraint of buffer space capability at least 269KBps.

[1] We describe "Bytes/sec" as "Bps", while "Bits/sec" as "bps".

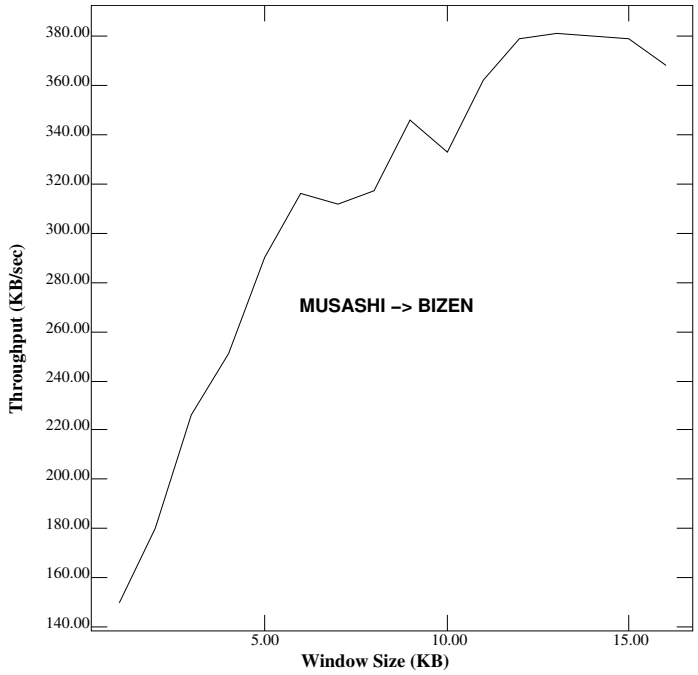


Figure. 5.3: Capability of Buffer Space

5.2 Comparison of CIP gateway and IP gateway

In order to verify advantages of CIP, we examined USP conversasions via an intermediate gateway. we compared two case of conversations, one is the case that the gateway supports CIP, while another is the case that the gateway does not understand CIP, that is, the former gateway works as a CIP gateway, the latter gateway works as a pure IP gateway.

In addition, we classified those conversations as following types:

- There is no congestion throughout the connection-path.
- There is a congested network between the gateway and the target. We simulated a congested network by reducing the input buffers of the gateway and the target extremely. We allocated only 2KB for input buffers in every IP receive thread at AIZU, BIZEN, and SURUGA. The reduced buffers result in high probability of packet loss and low throughput across the sub-network.

  Since it is difficult to simulate network traffic that is subject to stochastic processes such as *fractional autoregressive integrated moving-average processes*[ESWV94], the condition of congested sub-network goes on stationally until the end of measurements.

When there is no congestion throughout the connection-path, both figures of CIP gateway and IP gateway seem almost same (Figure. 5.5, Figure. 5.6). Transfer rate was distributed as

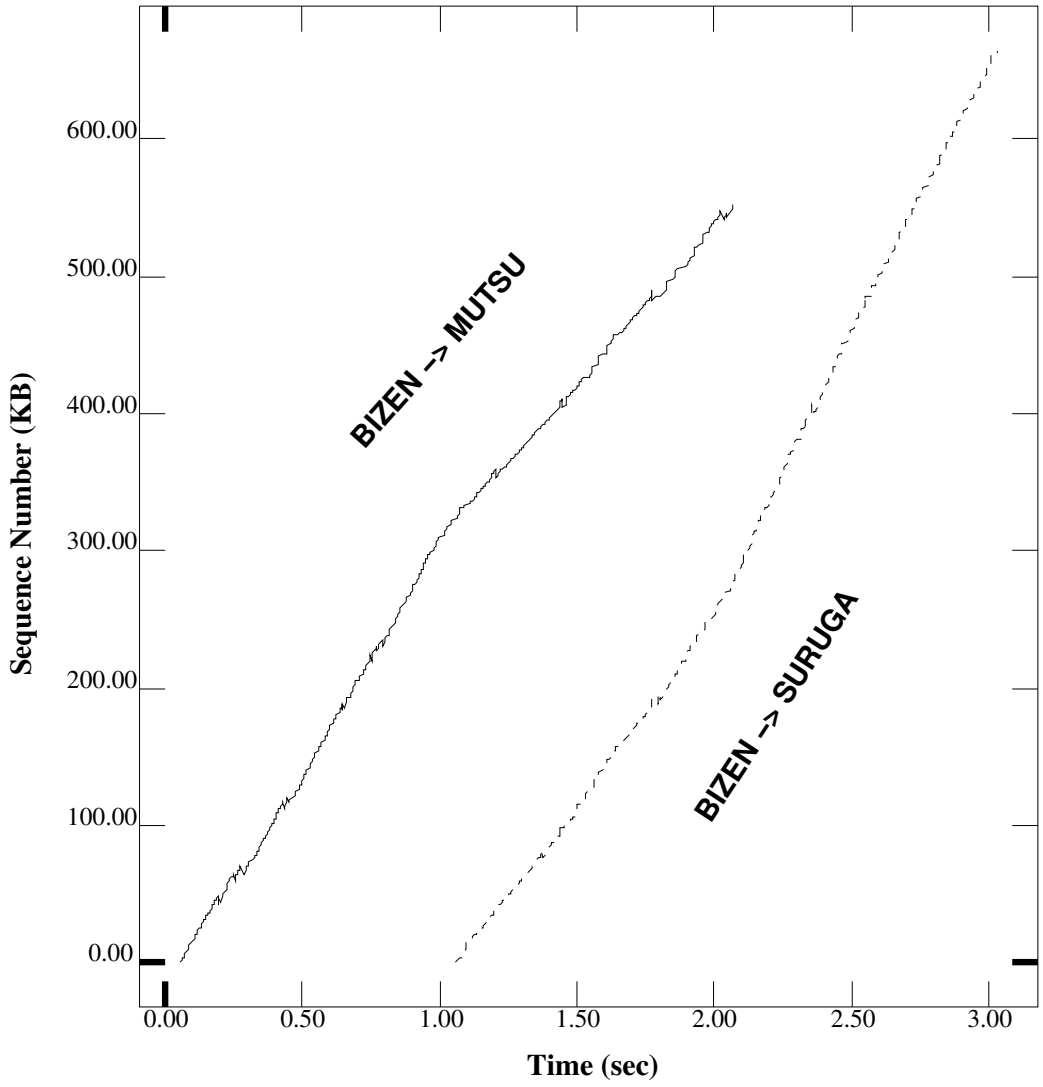


Figure. 5.4: Time Sharing Effects

Branch	0.0∼1.0	1.0∼2.0	2.0∼3.0[sec]
The former	327	221	
The latter		278	397
Total [KBps]	327	499	397

199KBps, 101KBps (the former), 133KBps, 201KBps (the latter) in the CIP → CIP → CIP connection.

In the mean time, Transfer rate was distributed as 212KBps, 131KBps (the former), 121KBps, 217KBps (the latter) in the CIP → IP → CIP connection.[2]

This result means there were no buffer that overflowed, and all of window advertisements were done well. In this case, advantages of CIP did not appear clearly, rather, IP excelled CIP in throughputs, about 7.4% advantageous over CIP. However, there is no more considerable difference in two figures between CIP gateway and IP gateway. Contrary, we think such difference should not appear on the only one-hopped internet, because it is TCP flow control mechanism that we employ in our protocol as the hop-by-hop flow control mechanism. Note that CIP-to-CIP communication via an IP gateway is almost equivalent to TCP-to-TCP communication via an ordinary IP gateway except for differences found in implementations.

In the mean time, when there is a congested network between the gateway and the target, obvious difference between CIP gateway and IP one is observed. (Figure. 5.7, Figure. 5.8).

Observed "jitters" on the figures indicate CIP retransmissions. Since the CIP gateway can detect the immediate congestion by itself, it always propagated back correct window advertisements directly, while the IP gateway went on propagating incorrect advertisements made by the target. This contradiction (or inconsistency) resulted in diminished throughput. Indeed, the retransmissions on the connection MUSASHI → (BIZEN) ⇒ AIZU, that is, CIP → IP ⇒ CIP, occupies 16% of the effective transfer rate. And the retransmissions of MUSASHI → (BIZEN) ⇒ SURUGA connection occupies 40% of the effective transfer rate similarly.

In the mean time, the CIP → CIP ⇒ CIP connection, MUSASHI → BIZEN ⇒ AIZU has only 4.5% retransmissions, and MUSASHI → BIZEN ⇒ SURUGA has only 5.7% retransmissions.

5.3 Rate Control

We experimentally attempted to apply rate control to CIP connections. Here we mention why we included the rate control mechanism to our protocol. First, it was easy for us to implement the XTP like mechanism in CIP software, for there was no additional primitive to be required for such purpose. Second, there was not any regulation mechanism of transfer rate in CIP protocol despite of the connection management capability of CIP. In other words, transfer rate of CIP connections was always unlimited although CIP was able to select suitable routes for individual connections. Third, we intended to demonstrate that CIP rate control works exquisitely, co-operating with flow control in the face of the transition of packet switch processing power.

[2] We depict congested network as "⇒", while not congested network as "→".

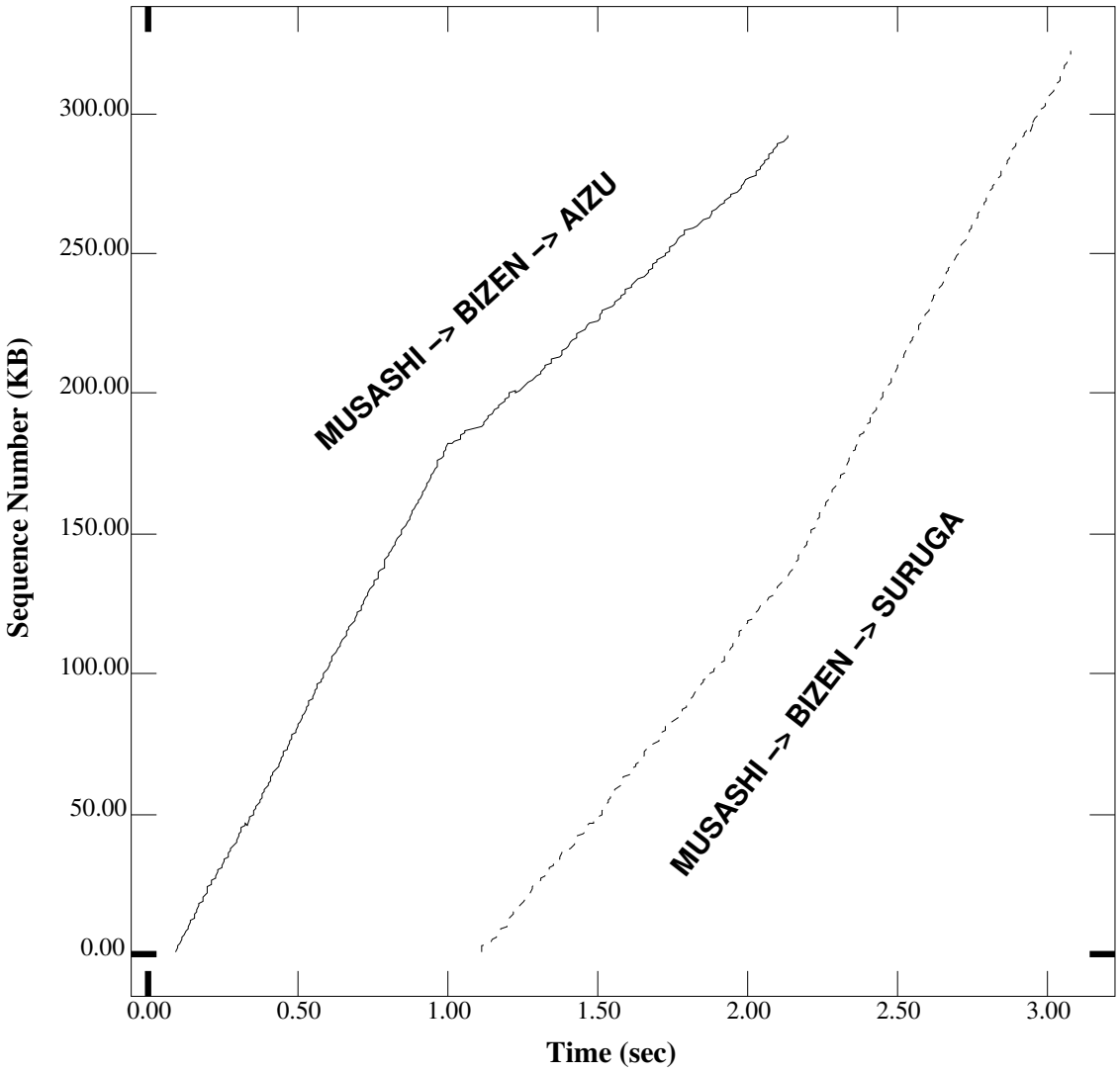


Figure. 5.5: CIP Gateway

CIP Gateway	0.0∼1.0	1.0∼2.0	2.0∼3.0[sec]
The former	199	101	
The latter		133	201
Total [KBps]	199	234	201

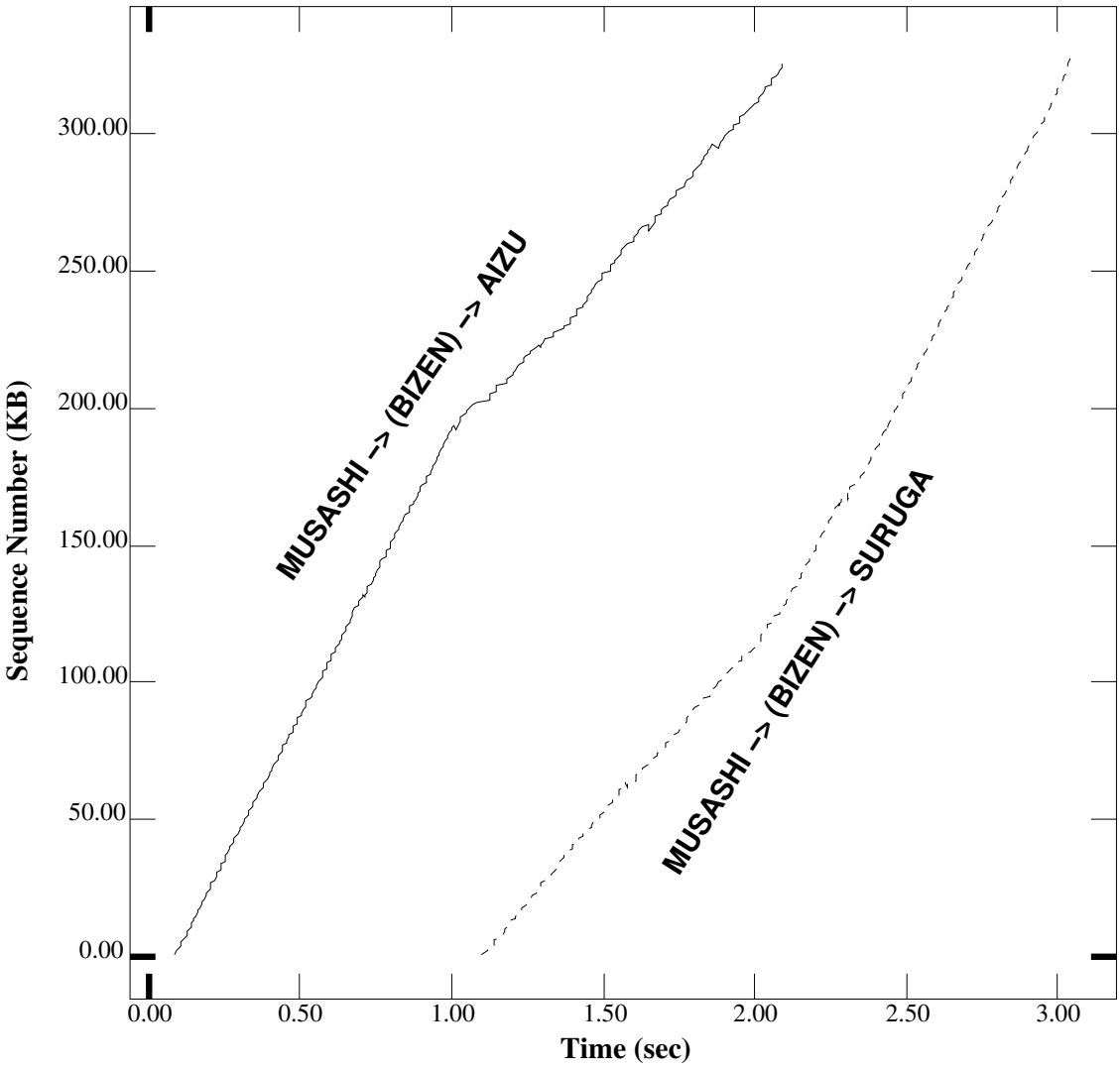


Figure. 5.6: IP Gateway

IP Gateway	0.0∼1.0	1.0∼2.0	2.0∼3.0[sec]
The former	212	131	
The latter		121	217
Total [KBps]	212	252	217

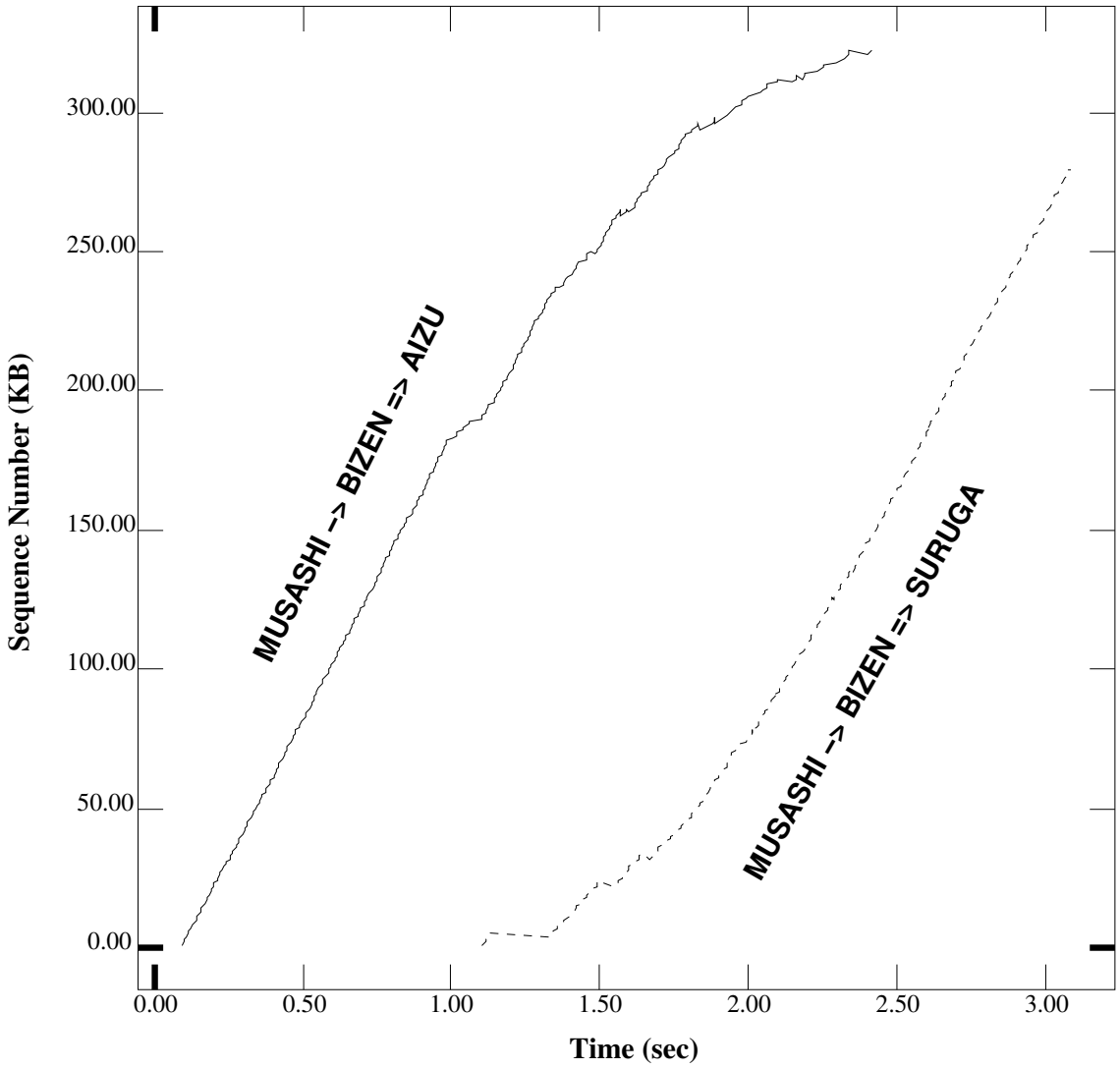


Figure. 5.7: CIP Gateway with Congestions

Retransmission rate [%]	The former	The latter
CIP Gateway	4.5	5.7
IP Gateway	16	40

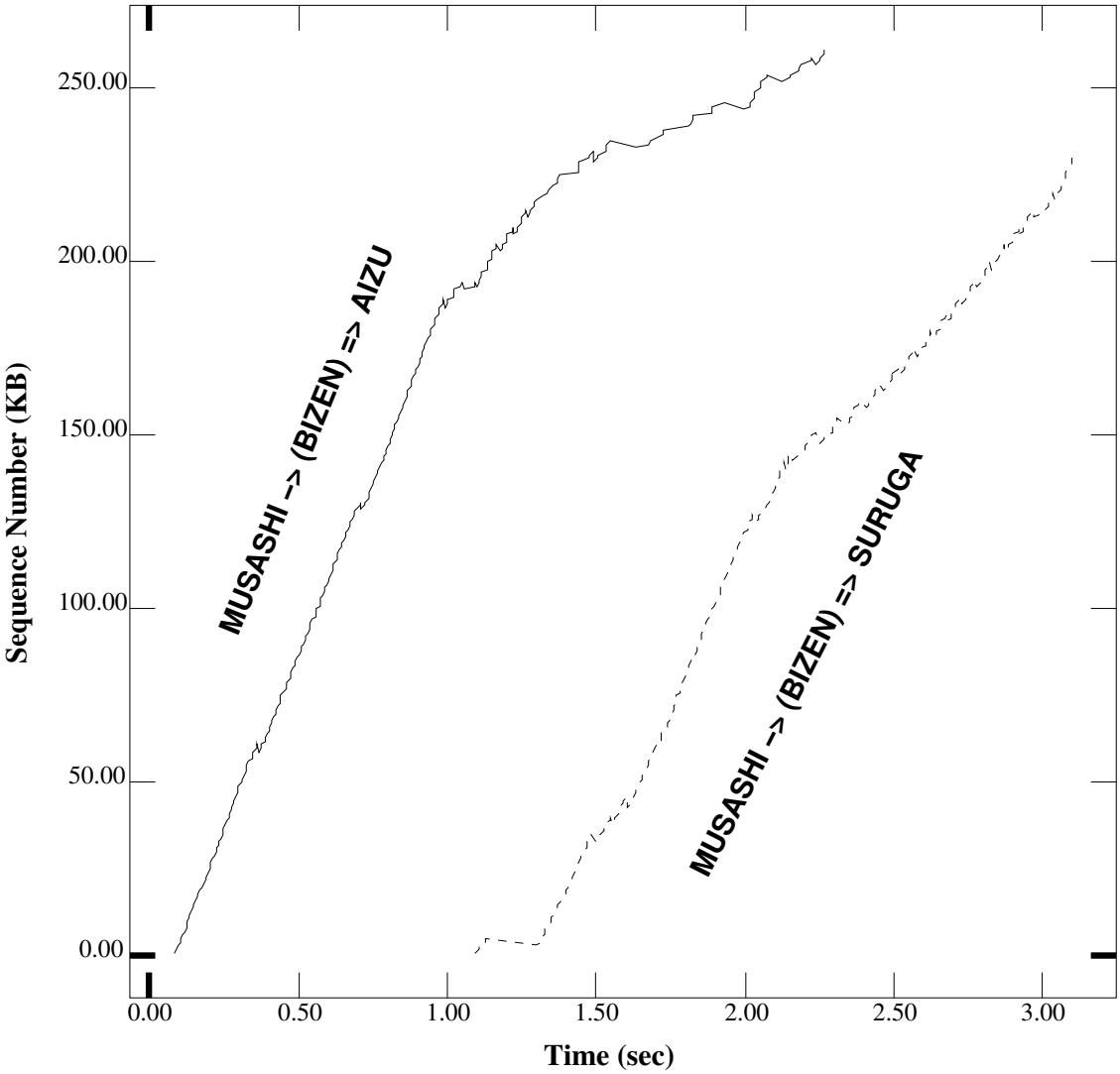


Figure. 5.8: IP Gateway with Congestions

Average throughput [KBps]	The former	The latter
CIP Gateway	138	140
IP Gateway	119	113

For such reasons, we measured several USP conversations each of which has following nature respectively:

- Rate controled conversations within capability of hosts. (Figure. 5.9)
- Rate controled conversations out of capability of hosts. (Figure. 5.10)

In the Figure. 5.9, one conversation is regulated at 100KBps transfer rate and has 4KB window (from MUSASHI to AIZU via BIZEN CIP gateway), while another is regulated at 150KBps and has 6KB window (from MUSASHI to SURUGA via BIZEN CIP gateway). Both conversations are not subject to the buffer space constraint. As a result, transfer rate was distributed as 96KBps, 89KBps (the former), 145KBps during the conversation (the latter).

In this case, we can observe the fact that rate control worked almost honestly. There was only 4% and 3% of relative error from specified transfer rate in the first and third area (discussed at "Simple Performance"), except for the second area, there was 11% of error, which was due to lack of processing power or collisions of packets.

In the Figure. 5.10, one conversation is regulated at 192KBps transfer rate and has 6KB window (from MUSASHI to AIZU via BIZEN CIP gateway), while another is regulated at 256KBps and has 8KB window (from MUSASHI to SURUGA via BIZEN CIP gateway). Both conversations are not subject to the buffer space constraint. As a result, the former obtained 188KBps in the first area, and 119KBps in the second area, the latter did 132KBps in the second area, and 253KBps in the third area.

In the first and third area, both conversations obtained almost all regulated rate, only 2% and 1% error, respectively. Since, however, the sum of required rate 448KBps exceeds the proved maximum rate 380KBps (from MUSASHI to BIZEN by any window size) in the second area, as a result, transfer rate was distributed as 119KBps (the former, 38% loss) and 132KBps (the latter, 48% loss). This error was due to lack of processing power or collisions of packets.

However, the reduction of their transfer rate is not important, rather, we should pay attention to the stable response found in both conversations against the immediate decline of packet switching bandwidth at one host. In general, without flow control or reservation of packet switching bandwidth in a certain way, rate control would result in a kind of panic around that host, since another host which was located previous to it went on injecting packets into it regardless of its shortage of switching power. In reality, (at least end-to-end) flow control prevents such situation, however, in our protocol, hop-by-hop flow control compensates for CIP rate control in the case that there are some hosts that are not able to guarantee pre-specified transfer rate, that is, upper bound of it.

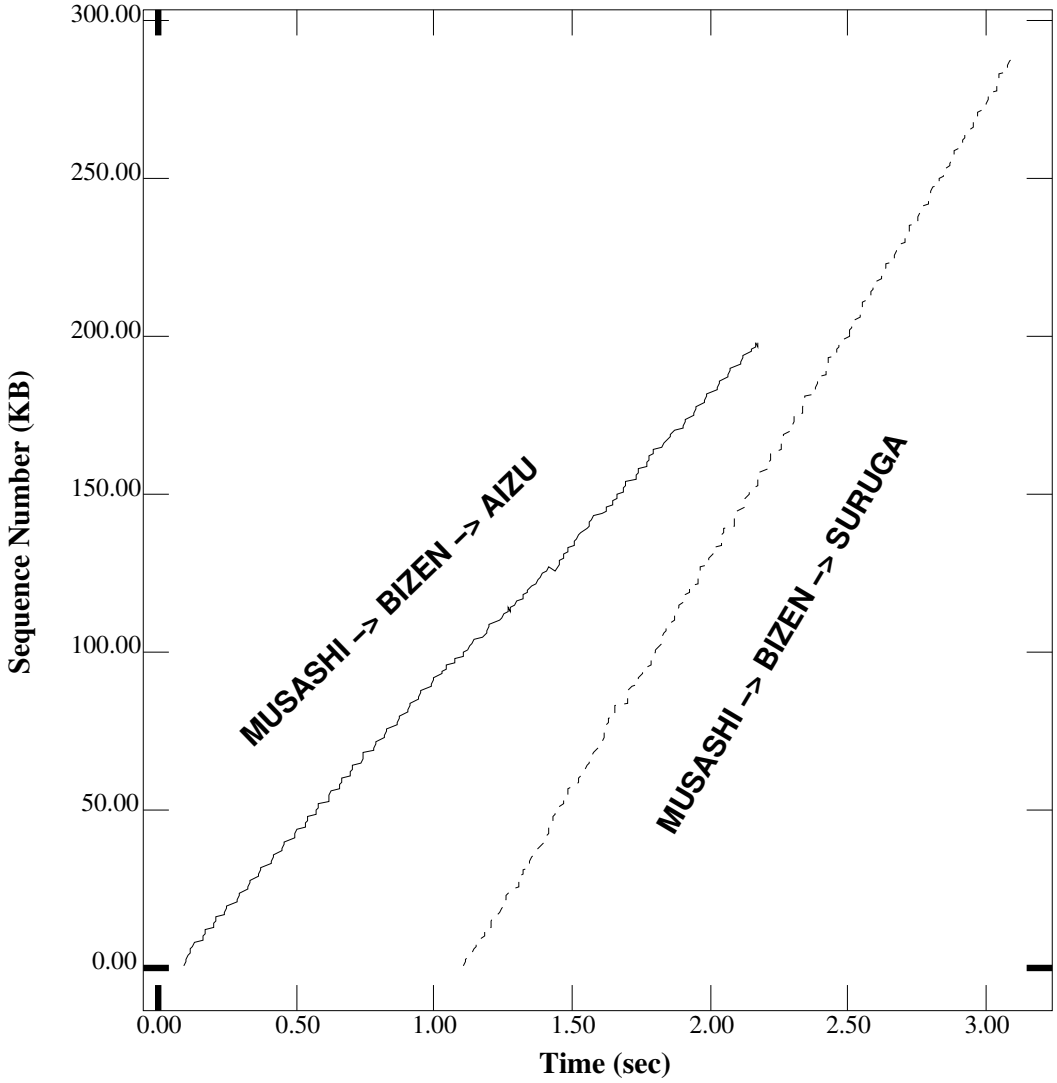


Figure. 5.9: Rate Control 100KBps, 150KBps

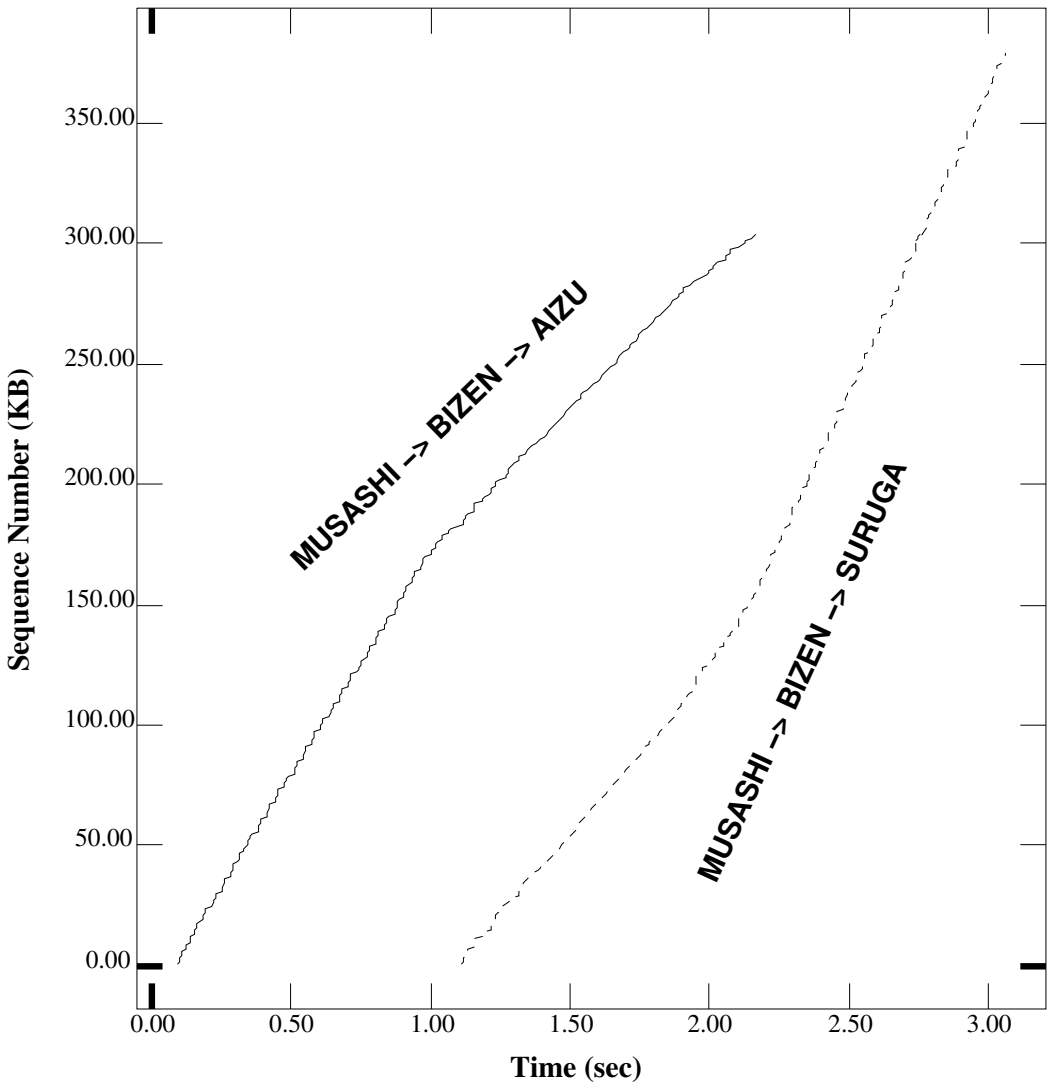


Figure. 5.10: Rate Control 192KBps, 256KBps

5.4 Discussion

5.4.1 Usability of CIP

As described previously, CIP showed several advantages in the face of congestions. Of course, this "congestion" is an experimental exaggeration, and it is truth that there is never stational congestion, however, we think this result shows CIP's preferable effects satisfactorily.

In summary, we mainly executed three control mechanisms, which the store and forward delivery mechanism enables, each of which has following meanings respectively.

- Hop-by-hop flow control
  Data flow synchronization. Hop-by-hop flow control essentially involves pre-allocation of buffers.
- Hop-by-hop error control
  Ensuring semireliability of CIP connections.
- Hop-by-hop rate control
  Regulation of transfer rate determining its upper bound.

Concerning the pre-allocation of buffers (refer to Figure. 5.3), we understood following affairs:

- We can also perform a kind of rate control with it. However, there are several problems if we employ it as our main rate control method, since if we reserve the same amounts of buffer for all CIP agents, and some pure IP area are included in the connection-path, the similar problems as resource reservation problems originally pointed out may occur. So we should not leave the pre-allocation method as it is.
- CIP divides end-to-end round-trip time into hop-by-hop ones, which leads to improved round-trip time as a whole.
- In other words, CIP can distribute effective buffers throughout the connection-path, for the large amounts of buffer at the target does not always lead to high throughput. The distributed effective buffers play an important role on multiple-hopped internet where uncertainness surrounding estimation of window size at the endpoints is apt to account for insufficiency of capability to fill up the connection-path with packets.

Succeedingly we discuss several problems not mentioned up to here clearly.

5.4.2 Overhead

CIP hop-by-hop flow control and error control results in principal overhead as compared with other connection-oriented internetworking protocols. As shown in Figure. 5.5 and 5.6,

CIP → IP → CIP connections excelled CIP → CIP → CIP ones in throughputs, about 7.4% advantageous over the latter. This result was seen on the one-hopped internet, however, if we make such experiment on a two- or more-hopped internet, would the overhead inclease linearly or exponentially? Since CIP connections construct a kind of data "pipe-line", where both injecting rate and ejecting rate of packets are ideally the same, the overhead does not seem to increase in such way, instead, it maybe persists even on multiple-hopped internet. Furthermore, buffer distributing effects possibly may compensate for CIP overhead on such a long connection-path.

5.4.3 Fragmentation

We have never mentioned about packet fragmentation in CIP. In fact, we did not implement such mechanism yet. However, we cannot help implementing it unless we yield up interconnectivity with IP, since there is always potential probability for datagrams to be fragmented by intervening IP gateways. In addition, since we employ the hop-by-hop flow control mechanism which is based on a series of sequence number agreed by a pair of hops, fragmentation and reassembly of fragmented datagrams should be done at every node whenever they are required.

5.4.4 Out-of-band delivery

In general, connection-oriented internetworking protocols have "out-of-band" delivery mechanism, which allows users to send urgent data regardless of state of connections. We think, however, the ordinary IP delivery mechanism should be substituted for such particular CIP urgent delivery method. To tell the truth, we have not determined whether CIP should equip out-of-band delivery by itself or not.

Chapter 6

Future Work

6.1 Optimization of CIP

6.1.1 Revision of congestion control strategy

In this paper, we used TCP flow control mechanism as CIP hop-by-hop congestion control mechanism. However, in the face of recent progress found in increase of network bandwidth, overhead of CIP hop-by-hop flow control seems relatively to become burdensome in the future. So we also incorporate mechanisms for rate control and pre-allocation of buffer space into CIP connection management as provision for the future extension of CIP. Using such mechanisms, we should revise our congestion control strategy in the near future. Rate-based congestion control[PK92] and the loss-load model[LWi93] are candidates for such strategy.

As compared with our congestion control scheme based on TCP flow control, the rate-based congestion control has several enhancements. First, it enables a cost-effective implementation for a multi-gigabit packet switch. It displays stable behavior for a wide range of traffic conditions, in addition, it does not cause oscillation of transfer rate. Second, it introduces "buffer setpoint" estimated using transfer rate means adequate amounts of buffers in a switch. It enables us to share limited buffers effectively.

In the mean time, the loss-load model introduces "loss-load curve" means the trade-off between the level of packet loss for a sender at a particular gateway as a function of that sender's offered load. That is, packet loss inside the network is a direct function of sender transmission rate, and senders have the responsibility of choosing their own transfer rate based on the loss-load trade-off curve provided by the network.

6.1.2 Separation of control messages and data

We have managed our protocol by a simple CIP option in the IP header. However, it has already turned out that the current CIP header includes unnecessary descriptors such as the "origin" field for established connections, while other descriptors such as "rate", "burst", and "buffer space" field are required anew. Because we have concentrated on the CIP congestion control mechanisms, we failed to arrange the header format, besides, connection management mechanisms skillfully. Involving re-arrangement of connection management procedure, we may have to separate control messages and data from each other.

We must optimize and specify CIP in detail, in particular, the lack of stream modifying mechanisms should be addressed in the first place. In addition, we intend to include the failure recovery mechanisms into CIP like ST-II, which detects the network or gateway failures and recover the broken stream automatically.

6.2 Optimization of our implementation

Succeedingly we have to optimize our implementation, especially, the network protocol development strategy such as layout of asynchronously communicating threads should be examined immediately. Then we need to profile CIP break-down in detail, and reasonable re-arrangement of thread layout should be done in response to the break-down.

OSPF and RSVP like connection setup protocol are also yet to be implemented. For our purpose, OSPF is quite attractive, and its implementation is required in any cost. While, RSVP like multicast support is eventually needed when CIP supports real-time flows in the future.

6.3 Simulation

In this paper, we generated synthetic "congestions" which did not seem stochastically realistic and compared "congested" network with not congested one, using it. Of course, we think we should simulate more realistic traffic and make CIP simulations there. For that reason, we have to analyze self-similar nature of Ethernet traffic described in [ESWV94] more closely. Otherwise, we should actually conduct experiments in existing environments which are thought that CIP is effective, such as packet radio network environment.

6.4 Resource management

Now that we can apply several control mechanisms for every stream of packets, following simple idea occurs:

- Network bandwidth reservation to some extent.
  To some extent that the stochastic process says it is necessary absolutely, we reserve the minimum amounts of network bandwidth.

- Resource management other than resource reservation.
  However, we manage other than the network bandwidth reservation, that is, CIP flow control with pre-allocation or cooperative sharing of buffer space and packet switch processing bandwidth, in order to guarantee some amounts of transfer rate, while permitting sharing limited network bandwidth among the individual streams.

We think a simple "automatic" resource configuration scheme, for exaple, automatic allocation and deallocation of required buffer space that is adequate for current transfer rate of one stream, becomes the first step for such future work.

Chapter 7

CONCLUTION

We proposed a new design of a connection-oriented internetworking protocol which we call "CIP" (Connection-oriented Internet Protocol). CIP has two major characteristics:

- CIP enables intermediate gateways to participate in congestion control by credit-based hop-by-hop sliding window flow control in which congestion can be directly detected and avoided.
- Existing IP gateways can be included in CIP connection-paths between end-points. So CIP has the advantage that it can coexist with IP networks, and reuse functionalities incorporated with IP.

We presented the implementation of CIP on Mach 3.0 as an application executed in the user address space. CIP succeeded in network layer level congestion control, retaining ability to coexist with IP. The results of our experiments show that CIP works well in the face of serious congestions. We think the new protocol design lead to advanced efficient utilization of existing network resources.

Furthermore, we demonstrated rate control including pre-allocation of buffer space, which gave us considerable hints lead to more effective congestion control, also including pre-allocation or cooperative sharing of packet switch processing bandwidth along each connection-path, enabled by connection-oriented internetworking protocols.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Mario Tokoro, who introduced me into the world of internetworking. I am deeply indebted to Mr. Atsushi Shionozaki, without whom this research would have never been possible. I would also like to thank Mr. Shunsuke Utsumi and Mr. Tatsumi Hosokawa for taking their time, checking drafts of this thesis paper.

I am grateful to all the members of the Tokoro Laboratory especially to the members of the Operating Systems and Networks Reserach Group: Mr. Naoyuki Sato, Mr. Shunsuke Utsumi, Mr. Seiji Murata, Mr. Mitsuaki Suto, Mr. Tatsumi Hosokawa, Mr. Kenichi Murata, and Mr. Atushi Shionozaki for their support and advice.

Above all, I would like to thank my parents and brother for their constant support and encouragement.

February, 1995

Bibliography

[Com91] Douglas Comer. *Internetworking With TCP/IP Vol I: Principles, Protocols, and Architecture Second Edition.* PRENTICE HALL, 1991.

[ESWV94] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, 2(1), 1994.

[HP91] Norman C. Hutchinson and Larry L. Perterson. The *x* - Kernel: An Architecture for Implementing Network Protocols. In *Transactions on Software Enginnering*, 1991.

[HTNH91] H. Tokuda, T. Nakajima, N. Ashida, and H. Arakawa. *Real-Time Mach 3.0 User Reference Manual.* Department of Computer Science Carnegie–Mellon University, May 1991.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *SIGCOMM '88*, 1988.

[KP87] Phil Karn and Craig Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. In *SIGCOMM '87*, 1987.

[Loe92] Keith Loepere. *Mach 3 Kernel Interfaces.* Open Software Foundation and Carnegie Mellon University, 1992.

[LWi93] Carey L. Williamson. Optimizing File Transfer Response Time Using the Loss-Load Curve Congestion Control Mechanism. In *SIGCOMM '93*, 1993.

[MN93] Chris Maeda and Brian N. Bershad. Protocol Service Decomposition for High-Performance Networking. In *SIGOPS '93*, 1993.

[Moy94] John Moy. OSPF Version 2. In *DARPA Internet RFC 1583*, 1994.

[NET87] *NET/ROM Version 1.3 Documentation.* Software 2000 Inc., 1987.

[PK92] Partho P. Mishra and Hemant Kanakia. A Hop by Hop Rate-based Congestion Control Scheme. In *SIGCOMM '92*, 1992.

[Plu82] David C. Plummer. An Ethernet Address Resolution Protocol. In *DARPA Internet RFC 826*, 1982.

[SJC92] W. Timothy Strayer, Bert J. Dempsey, and Alfred C. Weaver. *XTP: The Xpress Transfer Protocol.* Addison Wesley, 1992.

[SMMJ89] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley Publishing Company, Inc., 1989.

[Tan89] Andrew S. Tanenbaum. *COMPUTER NETWORKS SECOND EDITION.* Prentice Hall, 1989.

[Top90] C. Topolcic. Experimental Internet Stream Protocol, Version 2 (ST-II). In *DARPA Internet RFC 1190*, 1990.

[ZDE*93] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Schenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. In *IEEE Network*, 1993.

Appendix

In this appendix, we show several programs and codes used in practice either in CIP software or in USP client/server.

Smoothed Round-trip Time Estimation Method

```
 1 int             rttSec, rttNano, mSec, mNano;
 2 int             aveSec, aveNano, devSec, devNano;
 3 timespec_t      now,  diff;
 4
 5   /*
 6    * Now, the buffer 'buf' is acknowledged.
 7    * If the buf was not retransmitted,
 8    * we calculate the new smoothed round-trip time.
 9    */
10
11   now = *mappedClock;
12
13   /*
14    * calculate difference time between the buf's time stamp and 'now'.
15    */
16   timespec_diff( diff, now, buf->timeStamp );
17   mSec = (int)diff.seconds;
18   mNano = (int)diff.nanoseconds;
19   /*
20    * mSec, mNano means measured round-trip time.
21    */
22
23   /*
24    * M <- M - A
25    * (Average is scaled in factor 8. i.e., g = 1/8.)
26    */
27   mSec -= (aveSec >> 3);
```

```
28  mNano -= ((aveNano >> 3) + ( ((aveSec & 7) << 13) * 15259 ));
29
30  /*
31   * A <- A + gM
32   * (We eliminate floating point calculations.)
33   */
34  aveSec += mSec;
35  aveNano += mNano;
36
37  /*
38   * M <- |M|
39   */
40  if( aveSec > 0 ){
41    if( aveNano > 0 ){
42      if( aveNano > NANOSEC_PER_SEC )
43        aveSec++, aveNano -= NANOSEC_PER_SEC;
44    } else if( aveNano < 0 ){
45      aveSec--, aveNano += NANOSEC_PER_SEC;
46    }
47  } else if( aveSec < 0 ){
48    if( aveNano < 0 ){
49      if( aveNano < -NANOSEC_PER_SEC )
50        aveSec--, aveNano += NANOSEC_PER_SEC;
51    } else if( aveNano > 0 ){
52      aveSec++, aveNano -= NANOSEC_PER_SEC;
53    }
54  }
55
56  if( mSec > 0 ){
57    if( mNano > 0 ){
58      if( mNano > NANOSEC_PER_SEC )
59        mSec++, mNano -= NANOSEC_PER_SEC;
60    } else if( mNano < 0 ){
61      mSec--, mNano += NANOSEC_PER_SEC;
62    }
63  } else if( mSec < 0 ){
64    if( mNano < 0 ){
```

```
65        if( mNano < -NANOSEC_PER_SEC )
66          mSec--, mNano += NANOSEC_PER_SEC;
67      } else if( mNano > 0 ){
68        mSec++, mNano -= NANOSEC_PER_SEC;
69      }
70    }
71
72    if( mNano < 0 )
73      mSec = -mSec, mNano = -mNano;
74
75    /*
76     * M <- M - D
77     * (Deviation is scaled in factor 4. i.e. g = 1/4.)
78     * Note that there is no necessity g for A and g for D are the same.
79     */
80    mSec -= (devSec >> 2);
81    mNano -= ((devNano >> 2) + ( ((devSec & 3) << 14) * 15259 ));
82
83    /*
84     * D <- D + gM
85     */
86    devSec += mSec;
87    devNano += mNano;
88
89    if( devSec > 0 ){
90      if( devNano > 0 ){
91        if( devNano > NANOSEC_PER_SEC )
92          devSec++, devNano -= NANOSEC_PER_SEC;
93      } else if( devNano < 0 ){
94        devSec--, devNano += NANOSEC_PER_SEC;
95      }
96    } else if( devSec < 0 ){
97      if( devNano < 0 ){
98        if( devNano < -NANOSEC_PER_SEC )
99          devSec--, devNano += NANOSEC_PER_SEC;
100     } else if( devNano > 0 ){
101       devSec++, devNano -= NANOSEC_PER_SEC;
```

```
102     }
103   }
104
105   /*
106    * RTO = A + 2D
107    */
108   rttSec = ( (aveSec >> 2) + devSec ) >> 1;
109   rttNano = ( (aveNano >> 2) + devNano ) >> 1
110             + ((aveSec & 7) << 13) * 15259
111             + ((devSec & 1) << 15) * 15259;
```

USP Client

An USP client used in Chapter. 5 is composed as follows:

```
 1 /*
 2  * thread argument type
 3  */
 4 typedef struct {
 5   unsigned short   port;
 6   unsigned short   window;
 7   unsigned short   rate;
 8   unsigned short   burst;
 9   timespec_t       endTime;
10   int              strLen;
11   ip_addr_t        target;
12   char             *str;
13 } th_arg;
14
15 th_arg ThArg[ 4 ];
16
17 /*
18  * client thread
19  */
20 void ClientThread( th_arg *arg )
21 {
22   usp_t        *usp;
23   timespec_t   now;
```

```
24
25   /*
26    * Active open.
27    * (Error procedures are omitted.)
28    */
29   usp = UspConnect( arg->target, arg->port, arg->window );
30
31   for( ; ; ){
32     /*
33      * UspSend passes arg->str to CIP in the form of buffers.
34      */
35     UspSend( usp, arg->str, arg->strLen );
36
37     /*
38      * Client terminates itself at arg->endTime.
39      */
40     now = *mappedClock;
41     if( timespec_cmp( arg->endTime, now ) ){
42       /*
43        * Client terminates USP services normally.
44        */
45       UspClose( usp );
46
47       thread_terminate( mach_thread_self() );
48       /*
49        * never reached
50        */
51     }
52   }
53 }
```

USP Server

An USP server used in Chapter. 5 is composed as follows:

```
1 /*
2  * server thread
3  */
```

```
 4 void ServerThread( int *port )
 5 {
 6   usp_t        *usp;
 7   int          len;
 8   char         str[ 1024 ];
 9
10   /*
11    * Passive open.
12    * Server notifies USP of its listening.
13    * (Error procedures are omitted.)
14    */
15   usp = UspListen( *port );
16
17   /*
18    * wait for connection establishment
19    */
20   UspAccept( usp );
21   /*
22    * connection established
23    */
24
25   for( ; ; ){
26     len = 1024;
27     if( FALSE == UspReceive( usp, str, &len ) ){
28       /*
29        * data transfer finished
30        */
31
32       thread_terminate( mach_thread_self() );
33       /*
34        * never reached
35        */
36     }
37   }
38 }
```