

HELSINKI UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
TELECOMMUNICATIONS SOFTWARE AND MULTIMEDIA LABORATORY

VILLE HALLIVUORI

A CONCEPT FOR PROTECTING STATEFUL ROUTING PROTOCOLS

Master's Thesis
10th September 2001

Supervisor: Professor Teemupekka Virtanen
Instructor: Heikki Heikkilä, Ph.D.

Author:	Ville Hallivuori
Name of the Thesis:	A Concept for Protecting Stateful Routing Protocols
Date:	10th September 2001 Number of pages: 14 +96
Department:	Department of Computer Science and Engineering
Professorship:	T-110 Telecommunication Software and Applications
Supervisor:	Professor Teemupekka Virtanen
Instructor:	Heikki Heikkilä, Ph.D.
<p>IP as a network technology has been traditionally associated with best-effort data networks. IP is getting more popular within telecom networks. These usages require usually high reliability. Traditional telecommunication equipment utilizes protection for enhancing reliability, but traditional IP routers lack this feature.</p> <p>In IP networks routing protocols should be protected to ensure that packets are sent to the protected router regardless of component failures. Commonly used BGP-4 protocol must therefore be protected.</p> <p>This thesis investigates suitable methods for protecting BGP-4 routing sessions in protected IP routers. Overview of existing technologies for high availability networks is given. Both TCP and BGP-4 protocols are introduced. Several methods for protecting BGP-4 routing sessions are presented in detail. Their suitability is evaluated and the most suitable method is selected.</p>	
Keywords:	TCP, BGP-4, protection, routing

Tekijä:	Ville Hallivuori		
Työn nimi:	Menetelmä tilallisten reitityskäytäntöjen suojaamiseksi		
Päivämäärä:	10. syyskuuta 2001	Sivumäärä:	14 +96
Osasto:	Tietotekniikan osasto		
Professori:	T-110 Tietokoneverkot ja tietoturvallisuus		
Työn valvoja:	Professori Teemupekka Virtanen		
Työn ohjaaja:	FT Heikki Heikkilä		
<p>IP verkkoteknologia on perinteisesti yhdistetty laatutakeettomiin (best-effort) tietokoneverkkoihin. IP:n suosio televerkoissa on kasvussa. Perinteiset televerkot käyttävät varmennusta lisäämään verkon luotettavuutta. Perinteiset IP reitittimet eivät tue varmennusta.</p> <p>IP verkoissa reitityskäytännöt on varmennettava, jotta mahdollisista vioista riippumatta paketteja lähetetään varmennetulle reitittimelle. Siksi yleisesti käytetty BGP-4 protokolla täytyy varmentaa.</p> <p>Tämä diplomityö tutkii eri menetelmiä BGP-4 yhteyksien turvaamiseksi suojatuissa reitittimisissä. Työ luo katsauksen IP verkoissa käytettäviin korkean käytettävyyden teknologeihin. Useita metodeja BGP-4 reititysyhteyksien suojaukseen esitetään yksityiskohtaisesti. Menetelmien käyttökelpoisuutta vertaillaan ja käyttökelpoisin menetelmä nimetään.</p>			
Avainsanat:	TCP, BGP-4, suojaus, reititys		

Preface

This work was done as a part of a larger effort for selecting protection solutions for a carrier class router platform. I thank all people associated with this project and hence making this thesis possible.

I would like to thank my advisor, Heikki Heikkilä guiding this work and providing his insight to real world protection issues. Matti Hallivuori deserves thanks for helping with the language and for providing helpful comments.

I also thank Pasi Kinnari and Mika Silvola for helping to define the subject and scope of this thesis.

Ville Hallivuori

10th September 2001

Contents

Abstract	iii
Tiivistelmä (in Finnish)	iv
Preface	v
Terms and abbreviations	xi
1 Introduction	1
2 System architecture	3
2.1 Backplane	4
2.2 Control card	4
2.3 Line cards	4
2.4 Software environment	5
2.5 TCP/IP stack	5
2.6 BGP-4 daemon	6
2.7 Route database	7
2.8 Hardware protection	7
2.9 Routing protocol protection	7
2.10 Summary of system architecture	8
3 Transmission Control Protocol	9
3.1 TCP header	9
3.2 TCP state machine	10
3.2.1 Embryonic states	10
3.2.2 Connected states	11
3.3 Retransmit and flow control	12
3.4 Segmentation and re-assembly	12
3.5 Opening connection	13
3.6 Closing connection	13

3.7	Error behavior	13
3.8	Summary	14
4	BGP-4	15
4.1	Peer relationships	16
4.2	Topology	17
4.3	Route reflection	17
4.4	BGP-4 state machine	18
4.4.1	Before transport session	18
4.4.2	Negotiating BGP-4 connection	19
4.4.3	Full BGP-4 connection	20
4.5	Summary	20
5	Socket interface	21
5.1	Creating sockets	21
5.2	Establishing connections	22
5.3	Receiving and transmitting	22
5.4	Asynchronous I/O	22
5.5	Closing connections	23
5.6	Inheritance of sockets	23
6	Current solutions for high availability routing	25
6.1	Routing protocols	25
6.2	HSRP	26
6.3	Virtual Router Redundancy Protocol	27
6.4	Redundancy	27
6.5	Resilient transport	28
6.6	Summary	28
7	Protection	29
7.1	Concept	29
7.2	Applications	30
7.3	Protection in traditional environment	30
7.4	Protection in IP environment	31
7.5	Benefits of protection	31
8	Previous work on TCP protection	33
8.1	Protocol extensions	33
8.1.1	Graceful restart mechanism for BGP	33
8.1.2	Fault tolerant LDP	36

8.2	Wrapping TCP stack	37
8.2.1	North side wrap	38
8.2.2	South side wrap	38
8.2.3	Logger	39
8.2.4	Normal operation	39
8.2.5	Switchover	40
8.2.6	Performance	40
8.2.7	Requirements for applications	41
8.2.8	Disadvantages	41
8.3	Transparent replication	42
8.4	Summary	42
9	Requirements	43
9.1	Goals	43
9.2	Protected information	44
9.2.1	TCP state information	44
9.2.2	BGP-4 state information	45
9.2.3	Other protected information	45
9.2.4	Summary	45
10	Bulk transfer methods	47
10.1	Memory copying	47
10.2	Encoding data	48
10.3	Conclusions	48
11	Protection methods	51
11.1	Method 1: logical synchronization of TCP and BGP	51
11.1.1	The need for synchronization	51
11.1.2	Logical synchronization	52
11.1.3	Switchover	55
11.1.4	Insertion of new control card	56
11.1.5	Signaling diagrams	57
11.1.6	Advantages and disadvantages	60
11.2	Method 2: replicating TCP state	62
11.2.1	Transmitting TCP segments	62
11.2.2	Receiving TCP segments	63
11.2.3	Socket interface	63
11.2.4	BGP-4 synchronization	65
11.2.5	Replication	65

11.2.6	Switchover	66
11.2.7	Insertion of new control card	67
11.2.8	Signaling diagrams	67
11.2.9	Advantages and disadvantages	72
11.3	Method 3: replicating TCP and BGP-4 state	73
11.3.1	Transmitting TCP segments	74
11.3.2	Receiving TCP segments	75
11.3.3	Socket interface	75
11.3.4	Routing database interface	75
11.3.5	Replication	75
11.3.6	Switchover	76
11.3.7	Signaling diagrams	77
11.3.8	Advantages and disadvantages	82
11.4	Method 4: graceful restart mechanism for BGP	82
11.4.1	Switchover	83
11.4.2	Advantages and disadvantages	84
11.5	Method 5: terminating TCP sessions at line cards	85
12	Comparison of the protection methods	87
12.1	Selection of protection method	87
12.1.1	Minimal interruption to the traffic	87
12.1.2	Software version cross compatibility	88
12.1.3	Compatibility with third party routers	88
12.1.4	Suitability for other TCP based applications	89
12.1.5	Minimal changes to existing code	89
12.1.6	Minimal overhead	89
12.2	Selected method	89
13	Adaptation to the carrier class router environment	91
14	Conclusions	93

Terms and abbreviations

AMX A real time operating system.

ARP Address Resolution protocol, a protocol used for resolving a link level address from an IP address.

ASIC Application Specific Integrated Circuit.

AS Autonomous System, a routing domain under a single administrative authority.

ATM Asynchronous Transfer Mode, a cell based network technology.

BGP-4 Border Gateway Protocol 4, a routing protocol for inter-AS reachability information distribution. [24, 23]

BSD Berkley Software Distribution, a distribution of UNIX operating system.

CLI Command Line Interface, the management interface of choice for most routers.

CPU Central Processing Unit, processor of a card.

DMA Direct Memory Access, a method of transferring memory blocks without using system processor.

FreeBSD A free UNIX derived from original BSD UNIX.

EGP Exterior Gateway Protocol. A term that refers to any routing protocol that is used for exchanging routing information between autonomous systems.

HSRP Cisco Hot Standby Router Protocol, a protocol for clustering routers. [17]

HTTP Hyper Text Transfer Protocol, a protocol used for transmitting WWW pages.

IANA Internet Authority For Assigned Numbers, the authority that coordinates assignment of various identifiers related to internet protocols.

IBGP Internal BGP, a mode of BGP where it behaves as IGP.

IEEE Institute for Electrical and Electronics Engineers.

- IETF** Internet Engineering Task Force, the body making internet standards.
- IGP** Interior Gateway Protocol, a class of protocols that perform routing within single AS.
- IPsec** Internet Protocol security. A mechanism for providing security services for IP traffic.
- ITC** Inter Task Communication, a reliable method for sending messages between tasks in different cards.
- LDP** Label Distribution Protocol, a signaling protocol used in MPLS networks.
- LSP** Label Switched Path, a virtual connection between two or more label switching routers.
- MD5** Message Digest 5, a hash function.
- MIPS** Million instructions per second.
- MMU** Memory Management Unit, a CPU component that handles memory address translations, memory protection and memory access accounting.
- MPLS** Multi Protocol Label Switching, a layer 2.5 switching technology.
- MSS** Maximum Segment Size.
- NTP** Network Time Protocol. A protocol for synchronizing clock with remote server. [19]
- OSPF** Open Shortest Path First, a popular link state based interior routing protocol.
- PDH** Plesynchronous Digital Hierarchy, a transport technology that is used in traditional access networks.
- POTS** Plain Old Telephone System, the traditional telephone system.
- RIB** Route Information Base, a database for BGP-4 routing information. One BGP-4 router contains several RIBs.
- RIP** Routing Information Protocol, a simple IGP.
- RSVP-TE** An extended version of resource reservation protocol for traffic engineering. RSVP is used in MPLS networks for signalling.
- RTOS** Real Time Operating System.
- SDH** Synchronous Digital Hierarchy, a resilient transport technology used in traditional transport networks.
- SOCKS** A proxy protocol for TCP and UDP based applications. [16]
- switchover** Procedure where a protecting card assumes responsibilities of the protected card.

TCB TCP Control Block, a data structure containing TCP connection state variables.

TCP Transmission Control Protocol, a reliable transport protocol. [22]

TDM Time Division Multiplexing, a traditional method for multiplexing traffic.

VPN Virtual Private Network. In MPLS a virtual network formed over existing network using LSPs.

VRID Virtual Router ID, an identifier used to identify a single virtual router in VRRP protocol.

VRRP Virtual Router Redundancy Protocol, an IETF standard for clustering routers. [15]

Chapter 1

Introduction

In recent years IP based networks have challenged traditional telecommunication networks (networks oriented towards Nx64 services) in nearly all application areas. Although originally designed for best-effort data services, IP networks are nowadays used in many more demanding application areas, such as voice and video transfer. IP devices have historically been designed with best-effort mind set, and therefore IP routers are not as reliable as their telecom counterparts.

Migration from old telecommunication technologies toward IP based solutions has created a need for more reliable IP based devices. Customers expect that voice over IP applications provide the same reliability as the traditional telephone service provides. Also new application areas such as mission critical eBusiness applications and tele medicine require high reliability.

A common method of enhancing robustness of telecommunication devices is protection – a redundancy of hardware where in the event of failure a backup hardware can replace the failed component with negligible effect on the level of service users are experiencing.

This thesis investigates how BGP-4 routing protocol sessions can be incorporated to a protected IP router. The purpose of the BGP-4 protection is to ensure that hardware and software failures and upgrades of equipment and software do not affect routing connections. A failure in BGP-4 routing connection leads inevitably to temporary disbanding of the failed router from the network and therefore to loss of service. The IP router design used as the base of this study is an upcoming carrier class router platform. The purpose of this thesis is to develop a method that can be used to switch BGP-4 sessions to the backup hardware when the primary hardware fails.

The next chapter presents the architecture of the carrier class router platform. Then transmission control protocol (TCP), border gateway protocol (BGP-4) and socket interface are presented in detail. Chapter 6 presents current solutions for high availability routing. Chapter 7 introduces the concept of protection in detail. Previous work is then reviewed and requirements are set for evaluating various solutions. Chapter 11 presents protection methods in detail and evaluates their suitability against the requirements. Chapter 12 compares advantages and disadvantages of the presented protection methods. Finally conclusions are presented and a suitable protection method is named.

Chapter 2

System architecture

This study is a part of the carrier class router platform development activities. The router has a throughput in excess of several gigabits per second. The router combines both the normal IP routing and multiprotocol label switching (MPLS). The router has a huge number of interfaces and tunnel end points.

The router is designed to provide the level of reliability and redundancy commonly associated with traditional telecom equipment. All parts except the backplane are redundant.

The carrier class router platform has three main components: backplane, control cards, and line cards. The control card can be protected by adding an additional control card. Line cards can also be protected either by link level protection or by load balancing. The three main components are shown in the figure 2.1. In the figure the full mesh connectivity provided by the backplane is emphasized with multiple connectivity arrows. The number of line cards per backplane is not limited to four shown in the picture.

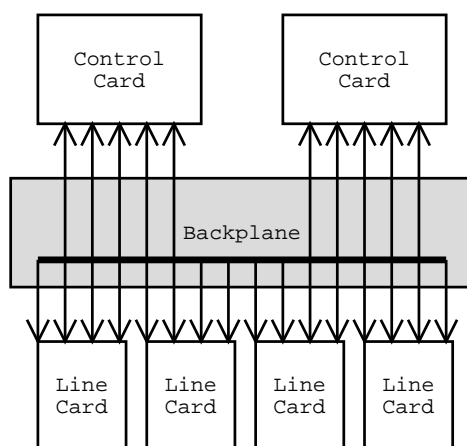


Figure 2.1: System architecture

The processor environment of the router is PowerPC. Typically the control card has a high performance G3 processor with more than 300 MIPS of computing power. Line cards have typically low

end PowerPC processors, each at least capable of 100 MIPS performance. Each control card contains at least 128 megabytes of RAM.

2.1 Backplane

All cards are connected to the backplane. The backplane is passive and does not contain a processor or ASICs. It is responsible for providing the gigabit level low delay point-to-point links between all possible card positions. Other functions of the backplane are to provide electricity and communication channels to support facilities (power units, cooling equipment, etc). The backplane contains hot swap slots for two control cards and for several line cards. Any card can be attached or removed on the fly.

2.2 Control card

The active control card is responsible for high level tasks. Its responsibilities include executing all routing protocols, executing MPLS related signaling protocols (LDP and RSVP-TE), maintaining management connections, and running other node wide tasks not directly related to forwarding.

Both control cards have their own processors. As software upgrade without disruption to the service is one important application of the protection, it must be assumed that software versions can differ between control cards. Control cards do not have a common processor clock source, and all clocking and time-keeping is independent. Real-time clocks are synchronized using NTP protocol. Although NTP protocol provides theoretical resolution of $1.1 * 10^{-10}$ s, poor local clock resolution, available NTP servers and large NTP polling intervals degrade the actual time keeping resolution. It must be assumed that the time difference between the active and the passive card can be several milliseconds. [19]

2.3 Line cards

Line cards combine switching logic and actual physical interface modules. Physical interfaces vary from gigabit ethernet to ATM connections. Line cards are responsible for performing actual forwarding and related tasks, such as collecting statistics for the control card. Although line cards are dependent on the existence of the control card for routing information and system maintenance functions, they are autonomous computers that can perform many tasks without the help of the control card.

All forwarding and switching is done by dedicated hardware on the line cards. This hardware consist mainly of ASICs and network processors. The hardware can perform regular switching, filtering, and bulk encryption without the assistance of the processor. The whole consisting of hardware and software participating in the actual forwarding is called as the forwarding plane. In a basic case forwarding does not require control card intervention – when the control card has programmed the

forwarding table to the forwarding plane and set up the ARP cache, no further intervention is necessary unless the routing table needs to be changed or the ARP cache needs to be updated. Obviously the control cards do collect various statistics from the line cards, but this has no effect on the actual forwarding.

Once programmed, the forwarding plane continues routing as long as there exist at least one functioning control card. Switchover does not stop forwarding or reset the forwarding tables. If both control cards malfunction, the forwarding plane automatically shuts down. Reasoning behind this behavior is that when there is no management interface, there should not be any forwarding – otherwise there could be situations where forwarding plane would forward wildly and network operator would have to go on-site to stop the forwarding.

2.4 Software environment

AMX real-time operating system is used on all cards. AMX is manufactured by a Canadian company Kadak. Perhaps the best known use of AMX is as the base of PalmOS used on PalmPilots. PalmOS is also a good example of versatility of AMX – PalmOS is actually AMX with static task configuration and custom interface library on top of it.

AMX offers real-time scheduling, message passing services, synchronization primitives and versatile memory management services. AMX is very small, and all aspects of it can be configured. AMX in no way limits what is done by the software.

All cards have capability for non-TCP based reliable inter-node communication. Such communication can be directed also as multicast for both control cards. This facility is here on referred as ITC.

2.5 TCP/IP stack

The TCP/IP stack is derived from a recent FreeBSD release. The TCP/IP implementation is single threaded and is executed as a separate thread. The software stack is only used for connections terminating on the card where the stack is running.

Most of the communication to the TCP/IP stack is through message passing. The three possible inputs for TCP stack are:

- TCP tick timer
- Network interfaces
- Socket calls

TCP stack always acts deterministically for these inputs. Only initial TCP sequence numbers are nondeterministic. First two of these events are messages and are processed by the dedicated TCP

thread. Socket calls are performed on caller's thread. Only one thread is allowed to be active on the TCP stack code. Multiple threads may be suspended on various checkpoints on the TCP stack.

TCP tick timer is a periodic message that initiates TCP stack's timer processing. As expiration of timer is through normal message passing interface, expired timer will not pre-empt running processing. TCP timers are handled as ordered lists which will be inspected when periodic tick message is received.

Network interfaces, such as ethernet drivers, send notifications of incoming packets through message passing. This causes execution of proper de-queuing procedures.

This architecture derives from the fact that FreeBSD had a single threaded stack.

2.6 BGP-4 daemon

BGP daemon performs all BGP-4 related routing operations. Like TCP stack, the BGP-4 daemon has several types of external events. These events are:

- BGP-4 tick timer
- Socket events
- Routing events
- Management actions

BGP-4 daemon maintains internal timers as ordered lists. These timers are inspected whenever BGP-4 tick timer expires. Tick timer does not pre-empt execution of BGP-4 daemon. BGP-4 uses timers for sending keep-alive messages that are used to verify connectivity to peers. Expiration of hold timer will cause connection to peer to be terminated.

Routing daemon uses nonblocking socket interface. Socket events are usually detected by select and mapped to a few internal events, such as socket writable or readable.

Routing events inform BGP-4 daemon of changes in routing table. These events are usually caused by other concurrently running routing protocols. For example if OSPF routing protocol learns some new route, BGP-4 daemon will receive route event and will then import the route found by OSPF. BGP-4 daemon sends routing events to route database about the routes it has learned from its peers.

There exist a proprietary method for synchronizing global routing database and other connection-less routing protocols. No further information of the proprietary method is presented in this paper, but it can be assumed that global route database is always up to date. It can also be assumed that route notifications will occur roughly in the same time on both cards.

Network management operations may change BGP-4 daemon's settings. These events could, for example, be adding or removing a new BGP-4 peer or creating a route reflector service.

BGP-4 daemon does not update the forwarding table (routing table). The responsibility of updating forwarding tables at IP stacks and at ASICs is or the other routing software that receives routing events from route database.

2.7 Route database

Route database will be automatically synchronized up-to-date when a passive card is booted on a node that already contains one active card. From the moment software is started, responsibility of maintaining up-to-date BGP-4 routing information (by sending events about BGP-4 route changes) becomes responsibility of the BGP-4 daemon running at the active card.

Implementation of the route database is not discussed in this paper, but we can assume that route entries from other protocols are always up-to-date.

2.8 Hardware protection

One of the control cards is active, and the protecting card is passive. If failure is detected on the active card, activity status is passed to the protecting card by a subsystem not discussed in this paper. There is, however, a mechanism that decides which of the cards is active and informs the software in question of state changes.

All traffic destined for control card is duplicated by dedicated hardware and reliably delivered to both the active card and the passive card.

Although the line cards are sometimes protected, this paper will not discuss mechanisms related to that area. Merely the fact of this possibility is acknowledged.

2.9 Routing protocol protection

Main routing protocol for external routing is BGP-4. Due to the use of MPLS and related virtual private networks, a single router can contain multiple instances of IGP daemon – all routing daemons are virtualised for each VPN. Usually routers have a capability to act as a BGP-4 route reflector. These factors ensure that the number of simultaneous BGP-4 connections can, and is likely to rise to thousands. Routing tables are expected to consume over 100 megabytes of memory.

The router has also non-BGP-4 TCP traffic. This traffic mainly consists of CLI (command line interface) management, node intercommunication and file transfers. During development phase also heavy load of debug traffic with proprietary TCP based trace protocol is transported. As both control cards originate different trace streams, these streams must not be protected.

A situation where active card changes is called switch-over. Such situation can be expected or unexpected. Expected switch-overs happen when control card is rebooted because of software update or card is removed for other reasons. Unexpected switch-overs occur if the active card becomes inoperable due to hardware or software failure.

To prevent BGP-4 peers from noticing switch-over, open TCP connections to BGP-4 peers must remain open, and all necessary routing state must be transferred to the passive card.

2.10 Summary of system architecture

The router platform is inherently modular and redundancy has been build in to the design. The architecture relies heavily in use of massive ASICs. Advanced hardware and software capabilities exist for building protection. Large amounts of CPU cycles, memory and communication bandwidth is available for the protection purposes.

Chapter 3

Transmission Control Protocol

This chapter presents the fundamentals of TCP. As TCP is a well known protocol, only the parts relevant for the discussion on this paper are presented thoroughly. A more detailed description of the TCP protocol is available at RFC793 and RFC1122. [22, 4]

Transmission Control Protocol, TCP, is a protocol that provides reliable connection based transport service over unreliable IP networks. Reliability means that TCP has re-transmit mechanism for lost segments and mechanism for dropping duplicate segments. Also mechanism for re-ordering out-of-order segments exists. TCP also performs rudimentary flow control.

As the notion of connection between communicating parties plays major role in TCP protocol, each connection has a state associated with it. Each connection must be created and terminated with special handshaking procedures. TCP connection can be seen as two way pipe between communicating parties – all data put in to the pipe appears sooner or later at the other end. Record boundaries are not preserved on TCP connections. All TCP segments belong to some connection, and only certain control flags are legal. TCP also provides multiplexing between communicating hosts – each machine has several ports for connections. A connection is identified by a 4-tuple (source IP, source port, destination IP, destination port). Individual TCP segments belonging to a connection are identified by sequence numbers. A segment with incorrect sequence number is dropped.

Although TCP is a connection based protocol, all state related to the connection resides on the communicating peers. Network is totally stateless. Network is not even aware of TCP sessions, as its only responsibility is to distribute IP packets.

3.1 TCP header

TCP header contains source and destination ports, sequence number, acknowledgment number, some flags, window advertisement, checksum, urgent pointer and some options. The exact arrangement of these fields has no relevance for us.

Flags of TCP header have following meanings:

ACK Acknowledgment field is valid. Used for acknowledging received segments.

FIN Flag indicating half-duplex closure of connection.

PSH Flag for forcing buffer flush.

RST Flag for resetting connection.

SYN Synchronize. Used for initiating new connections.

URG Flag indicating out of band data.

Options are not particularly relevant for the purposes of protection. The most common option is MSS, which is used to negotiate maximum segment size that both peers support. Other common options are used to enable selective acknowledgments and keepalive timers [18]. The only impact these options have on protection is to add some variables and buffers to the set of protected state information.

3.2 TCP state machine

The figure 3.1 shows the TCP connection state machine as presented in RFC 793 with corrections from RFC1122. There exist one state machine for each TCP connection. [22, 4]

A state where connection does not exist is represented on the connection state machine as the state CLOSED. Practically this means that the TCP stack has no state at all for those connections.

3.2.1 Embryonic states

Term embryonic is used to refer the states where initiation of connection is in process, but connection has not fully been formed.

The states are:

LISTEN The state in which host accepts active connection attempts. This state is the result of a passive open call (listen call of socket interface).

SYN SENT The state in which a host is, if it has initiated an active open attempt. This state is the direct result of a connect call of socket interface.

SYN RCVD The state in which the host waits the confirmation of establishment of a newly negotiated connection. The confirmation comes in the form of a packet acknowledging the sent SYN,ACK packet.

Embryonic states are generally not interesting from the viewpoint of protection, as no connection has yet been committed.

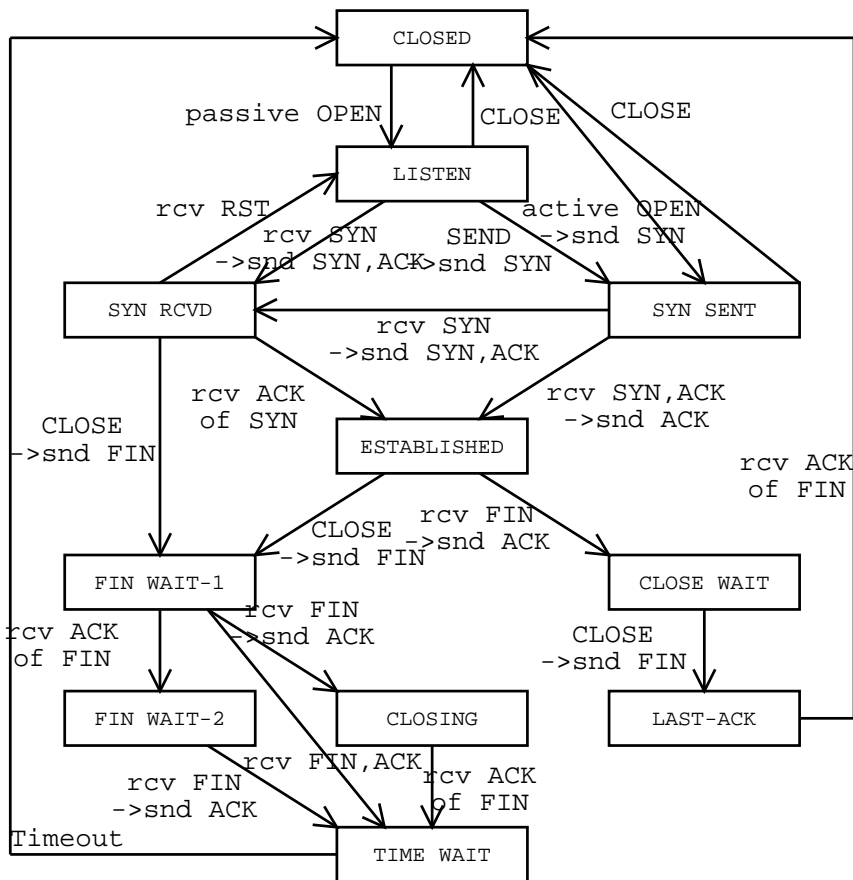


Figure 3.1: TCP connection state machine

The move from SYN SENT and SYN-RCVD state to ESTABLISHED state does not exist in normal stacks with BSD traits. Instead a new connection is created in ESTABLISHED state and old state machine is returned to LISTEN state. This behaviour is consistent with the protocol and is normal for stacks that use socket interface. [25]

3.2.2 Connected states

Connected states contain open communication channel at least to one direction. Connected states are:

ESTABLISHED The normal state of connection. Communication channels to both directions are open.

FIN-WAIT-1 The state in which a host has requested the transmit channel for this connection to be closed and waits for an acknowledgment from the remote peer.

FIN-WAIT-2 A Host has closed the transmit channel for this connection. The receive channel remains open.

CLOSE WAIT A state in which the remote host has closed its transmit channel for this connection. Our transmit channel remain open until we close it.

Usually peer closes its transmit channel when it has nothing more to send. These situations are common in everyday protocols, such as FTP and HTTP. These situations do not normally occur in routing protocols, such as BGP-4.

3.3 Retransmit and flow control

The TCP flow control has two purposes: to prevent a fast machine overwhelming a slow machine, and to prevent congestion on the network. TCP uses sliding window protocol, and receiver advertises on each acknowledgment packet, how much data it is willing to receive.

The first goal is accomplished easily: receiver only advertises as much window as it is willing to receive. To avoid too small segments (situation known also as Silly Window Syndrome), the receiver will advertise window only in large increments.

The second goal is accomplished by having additional congestion window. When segments are lost, this congestion window is halved. When segments are successfully acknowledged, the congestion window is slowly increased. The transmit window is never larger than congestion window, even if receiver had advertised larger one. As congestion manifests itself as packet loss, this mechanism reacts to the effects of congestion and therefore mitigates it.[21, 7]

TCP automatically calculates round trip time estimates from sent packets. Round trip times are used to calculate resent timers. This allows retransmit time to be appropriate for all types of networks and network loads. If segment is lost, round trip estimate is increased. Usually Karn's algorithm is used for calculating round trip estimates.[14]

All data and control segments are re-sent if no acknowledgments are received. Acknowledgments are unnumbered and are not acknowledged.

3.4 Segmentation and re-assembly

TCP stack segments outgoing stream based on the advertised window and the congestion window. TCP implementation tries to collect data to make segments as large as possible, within limits of MTU and window.

TCP acknowledgment indicates the sequence number of the next unreceived octet of data. That means that when TCP receives packets, it acknowledges only continuous segments – if the middle one of three segments is lost, only first is acknowledged. This behavior limits complexity of re-transmit algorithm. This behavior can be utilized in protection, as protocol allows TCP implementation where re-assembly buffer never contains acknowledged data (all data can be moved to application level buffers before sending acknowledgment).

3.5 Opening connection

TCP connection can be opened with two possible ways. First is the case where a server performs a passive open (application's indication for stack of willingness to accept incoming connections) and client performs an active open. The other rarely used way is simultaneous connection where both connection parties perform active open simultaneously. Both situations can occur on BGP-4 connections, as all routers are equal: every router is both client and server.

A connection is opened with a three way handshake. The name comes from the fact that three TCP segments are sent between parties. The respective packets contains flags SYN, SYN-ACK and ACK. If the handshake is successful, connection state machine moves to state ESTABLISHED. During the first two packets of the handshake both parties choose their initial sequence numbers. Sequence numbers are nowadays chosen with strong random number generators to avoid easy TCP connection hijacking[3].

3.6 Closing connection

A newly created TCP connection is always full-duplex. Either of communicating parties may at any time close outgoing channel just by sending a packet with FIN flag on. The TCP connection is maintained until both peers have closed their transmit channels. Closing connection channel only affects to sending data – acknowledgment segments are unaffected. Connection is also terminated, if acknowledgment is not received despite of repeating retransmit. TCP is quite persistent, and connection has to be down for several minutes to this timeout occur.

3.7 Error behavior

If packet with RST flag is received at any state where a TCP connection is open to at least one direction (states ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2 and CLOSE-WAIT) and sequence number of the reset is in acceptable range, the connection is terminated immediately. On the other states reset cancels connection opening. When packet with RST flag is received, a response is never send. RST is used only on error situations and when far end is not willing to receive connections.

If an incoming segment contains unacceptable sequence number, the TCP stack must send an ACK packet with the next expected sequence number. Nowadays reasonably implemented stacks avoid this behavior as the specified behavior can cause ACK storms, if malicious attacker manages to desynchronize TCP connection. Commonly observed behavior is to terminate connection with RST, if a segment with too large sequence number is received. Behavior exhibited when a TCP segment is received with an unacceptable sequence number varies between TCP implementations and therefore no general rule may be derived.

As many stacks do not closely follow the RFCs, or any other common behavior, this behavior can not be trusted to be consistent.

If incoming segment with incorrect ACK number is received, the segment is dropped. If acknowledgment refers to something that has not yet been sent, a packet with ACK flag and correct sequence number must be sent.

All segments with incorrect sequence numbers are dropped. All segments with incorrect checksum are ignored.

3.8 Summary

TCP is a complicated protocol that has a complex behavior. TCP offers reliable connection based communication service. Although use of TCP is simple for application, at network layer it involves complicated data exchanges. Maintaining TCP connection requires detailed knowledge of the state of the TCP connection state machine and associated variables.

The TCP connection state machine utilizes timers and therefore state transitions can occur without outside input.

Chapter 4

BGP-4

Border Gateway Protocol 4, BGP-4, is an exterior gateway routing protocol. Its purpose is to transmit reachability information between autonomous systems. BGP-4 protocol is successor to EGP protocol and is widely used for routing in Internet backbones. [24, 23]

BGP-4 is a link state protocol. BGP-4 uses TCP as its transport protocol. BGP-4 connections are always terminated to port 179 on both ends. BGP-4 supports classless inter-domain routing. BGP-4, unlike its predecessors, allows arbitrary topologies, and always produces loop free routes. Support for complex topologies is essential, since nowadays Internet contains several more or less competing backbones.

Although BGP-4 is mainly used for inter-AS routing, it can also be used within single AS as IGP. Internal BGP (IBGP) offers several advantages over traditional IGPs. Most important advantages are possibility to control exit point from AS and always consistent picture of AS for external BGP-4 speakers.

Autonomous systems are administrative areas, and commonly one organization has one AS. Each autonomous system, AS, is identified by a unique AS number. AS numbers are assigned by regional registrars of IANA.

Each BGP-4 router has three conceptual route information bases: Adj-RIBs-In, Loc-RIB, and Adj-RIBs-Out. Abbreviation adj is shorthand for word adjacent and indicates that a RIB bearing it contains information specific to a single neighbour. Instances of both adjacent RIB exist for each connected BGP-4 peer. Adj-RIBs-In contains all routes received from other routers. The Loc-RIB contains all the routes that the router has selected from Adj-RIBs-In for local use. Selection of routes is based on local routing policy, which is used to calculate degree of preference for each of the routes. Local policy is used to construct Adj-RIBs-Out from Loc-RIB and from routes learned from other routing protocols, such as from OSPF. Routes in Adj-RIBs-Out are advertised to other peers.

Policy routing is one of the more interesting features of the BGP-4. The purpose of policy routing is to bring real-life considerations to route selection. For example policy routing can be used to allow all traffic to and from our AS, but only allow transit traffic for paying customers. In similar fashion one could use policy routing to avoid our traffic from going through competitor's AS.

BGP-4 uses hop-by-hop paradigm: router may only advertise routes it would use. Each route advertisement contains list of autonomous systems the advertisement has traversed. The AS path information is used for avoiding routing loops and for policy routing. Each route advertisement also contains path attributes associated to the route. In addition to AS path there are some other path attributes, such as next hop address and the origin of the route prefix. Some attributes are mandatory, but BGP-4 also allows optional path attributes. Additional attributes can be defined, and there exists defined behavior for dealing with unknown attributes.

BGP-4 tries to minimize routing related network traffic, as routing table sizes and number of peers are huge on the core routers. To further this end, BGP-4 exchanges routing databases only once, and after that sends incremental updates when topology changes. Unlike in OSPF or in RIP, routes are not periodically advertised. Instead a special keep-alive message is used to test connectivity. If keep-alive messages are not received at pre-determined interval, or if TCP connectivity is lost, all routes learned from the missing router are removed. As rapidly changing topologies could cause immense amount of routing traffic, BGP also instantiates limits on minimal time between route advertisements concerning routes to single destinations. Usually route flapping (rapid cycling between connectivity and lack of connectivity) has a penalty delay up to 30 minutes configured to be imposed for BGP-peers that lose connectivity. BGP-4 also reduces route advertisement sizes by using route summarization. Route summarization reduces routing table sizes by combining several routes as a single route with less specific prefix. [12, 26]

BGP-4 supports user-defined authentication algorithms. Authentication facilities offered by BGP-4 protocol are not supported by large manufacturers (Cisco, Juniper, Riverstone) and therefore they are rarely used in real-world. Instead MD5 based authentication of TCP segments is used. The method is not strong, but is considered robust enough. [10]

4.1 Peer relationships

Unlike many IGPs, such as OSPF and RIP, BGP-4 does not have a peer discovery mechanism. Instead all BGP-peers must be manually configured for each BGP-4 speaker. Peer discovery is not vital, nor always feasible for BGP-4 protocol – connections between ASs are rarely multicast capable LANs. Implementing peer discovery would also require non-TCP based solution.

TCP is used as transport protocol, because reliable transfer of messages much larger than MTU is often required. As TCP is used, peer relationships always contain two parties. Neither of the parties is server in the traditional sense of TCP application. Instead both parties try to open a TCP connection to the other and at the same time accept new connections from peers. BGP-4 protocol contains deterministic decision process for closing the extra connection that can occur if both parties manage to open connection.

When TCP connection is opened, both parties send BGP OPEN message. Open message identifies AS of the sender, values of hold time, BGP identifier of the sender and a number of optional

parameters. As with all other messages, MD5 authentication is often used to verify the integrity and the origin of the messages.

The hold time indicates the maximum time that may elapse between packets received from the peer. Hold times can be different to both directions. The purpose of the hold time is to allow configuration accuracy of link state detection – with small hold time the peer has to send often keepalive messages, whereas with large hold time keepalive messages are sent rarely, if ever. Obviously quickness of detecting failed links depends on how short the hold time is.

Optional parameters can be used to extend the protocol in a manner that does not cause incompatibilities with other BGP-4 peers. RFC1771 defines only one option, authentication information, which can be used to select authentication algorithm. The option is rarely used.

If parameters specified in OPEN the packets are acceptable for both peers, the connection is confirmed by both parties by sending a keepalive message. Thereafter content of Adj-RIBs-Out is transferred to the remote peer. Route information is transferred in UPDATE message. After initial synchronization UPDATE messages are only sent if route changes are detected. A route may be withdrawn at any time, but frequency of route change advertisements is limited. Keepalive messages are sent, if specified hold time would elapse without UPDATE messages.

4.2 Topology

External peers (peers that belong to different ASes) must share common network or point-to-point link. Traffic between external peers must not depend on IGP. Single AS can, and usually has, several BGP-4 speakers. BGP-4 requires that all BGP-4 speakers within AS must present consistent picture of routes offered by the AS. In practice this means that BGP-4 speakers must wait until IGP routes converge before sending route advertisements of new routes.

4.3 Route reflection

Internal BGP-4 has a rule that no route learned by IBGP must be distributed to other BGP speakers in the same AS. In practise this rule means that IBGP speakers must have fully connected mesh of peer relations. Such connection mesh is possible for small AS, but as the number of needed connections grows squarely as a function of the number of the IBGP speakers within AS, this presents a huge problem for larger AS.

Route reflector is a special IBGP node that is allowed to distribute routes learned by IBGP. Route reflector has two kinds of peers: clients and non-clients. Clients are BGP-peers that are aware of the route reflector, and instead of full mesh to other clients, have connection only to the route reflector. Non-clients are BGP speakers that are not aware of the route reflector and have full mesh between all other non-clients. Non clients have also connection to the route reflector. The route reflector forwards routes from non-clients to all of its client. The route reflector also forwards routes from clients to all

other clients and non-clients. In a way a route reflector replaces full mesh of $n(n - 1)/2$ connection with $n - 1$ connections. Although it means that the number of connections per router is small, the number of connections per route reflector is significant.

Although routing messages pass through the route reflector, normal IP traffic usually does not. Route reflectors do not change next hop path attribute, and therefore no next hop information is lost on route reflection.

A group of clients and a route reflector is called a cluster. AS can have several clusters. When multiple clusters exist in single AS, peer relations between route reflectors form a fully connected mesh. The arrangement is analogous to the backbone area of OSPF protocol. A single cluster may have more than one route reflector.

Routing loops are avoided by defining new path attributes `ORIGINATOR_ID` and `CLUSTER_LIST`. Route reflector always defines `ORIGINATOR_ID` to be router id of the router that originated the route. If a router receives a route with `ORIGINATOR_ID` equal to its router id, the route is ignored. Route reflector always adds its cluster id to `CLUSTER_LIST` path attribute. If a reflector receives a route in which `CLUSTER_LIST` attribute contains router's cluster id, the route is discarded. [2]

4.4 BGP-4 state machine

BGP maintains single state machine for each peer. It should be observed that although there momentarily can be two transport connections between peers, there is always one state machine per peer. The BGP-4 state machine is presented on figure 4.1. In the interest of clarity not all self-transitions have been drawn.

4.4.1 Before transport session

Initially BGP-4 connection state machines are initialized to be in state idle. Connection state machine is on the idle state only initially and after serious uncoverable error. When management application brings connection up, peer state machine moves to state connect. Management application does periodically bring connections up from idle states. This periodic behavior restores connectivity after serious errors.

State connect is the state where BGP-4 daemon has requested TCP stack to actively open connections. Also passive connections are accepted. If either of active or passive connection attempts success, peers state machine moves to the state opensent. If connection attempt fails, peer state machine moves to the state active. Connection attempt is repeated with predetermined intervals.

In the active state BGP accepts connection attempts but does not try to actively connect. When predetermined interval has passed without connection, peer state machine assumes state connect again. Purpose of the active state is to limit frequency of connection attempts.

If a TCP connection is established in either active or connect state, BGP open message is sent to the remote peer. The peer state machine assumes the state opensent. If the connection is not from the

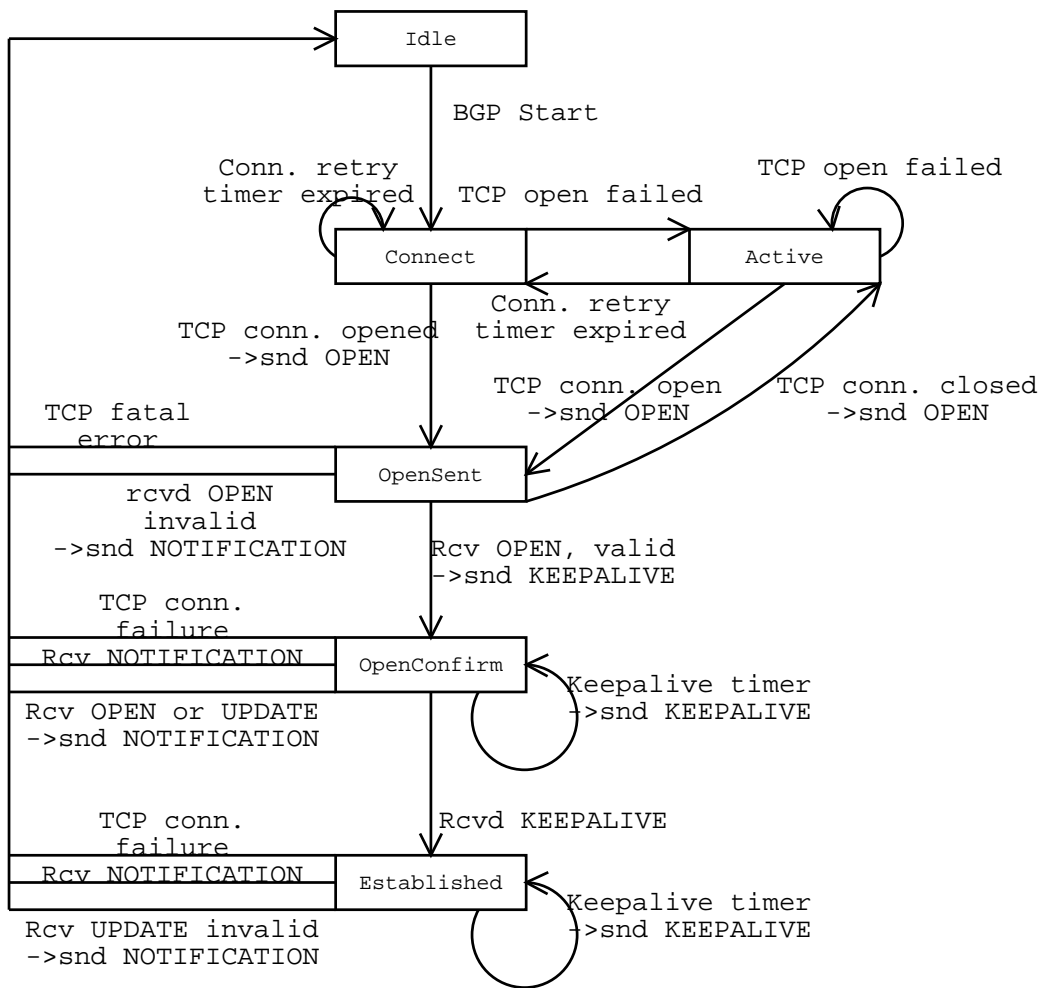


Figure 4.1: BGP4 peer state machine

expected address, the peer state machine is reseted to the state active.

Although in theory this procedure always guarantees connection, in reality there appears to be problems with several implementations. Slight incompatibilities in TCP and BGP-4 stacks and unfortunate timing may cause unexpected loss of connections. It might take several tries and minutes before connections are established between routers of different brands.

4.4.2 Negotiating BGP-4 connection

During the transition to the state opensent, the BGP daemon sends BGP open message over the newly opened TCP connection. A BGP open message identifies the sending peer and transmits options to be negotiated.

If a notification message is received from a remote peer or a fatal TCP error occurs, the peer state machine assumes the state idle. If the open message is received with unknown options or with

incompatible connection parameters, a notification message is sent and the state machine assumes the state idle. A widely used BGP-4 extension specified in RFC2842 allows advertisement of optional capabilities without dropping connection when unknown option is encountered [5].

If transport is closed, the peer state machine assumes the state active. When an open message is received, the receiver must first check if there already exists a connection to router with the same BGP identifier. If such connection exists, and the BGP identifier of the receiving router is less than sender, the existing connection is closed. Otherwise the new connection is closed. The acceptable open is acknowledged with keepalive message. The state machine assumes the openconfirm state.

In the openconfirm state the BGP-4 daemon waits for the confirmation of the connection from the remote peer. A connection establishment can only fail due to a network error or because a remote peer considered our open message invalid.

4.4.3 Full BGP-4 connection

When keepalive message is received in the openconfirm state, connection to the peer is considered to be established. In the established state three types of messages are exchanged: updates, keepalive messages and notifications.

Update messages transmit route information with associated path attributes. Notifications are used to inform of fatal errors. Keepalives are used to test connectivity whenever there is a long idle interval in regular update traffic.

4.5 Summary

Although BGP-4 is a highly versatile and complex routing protocol, its connection related portion is simple due to the use of the TCP as the transport protocol. BGP-4 withdraws all routes learned from a remote peer, if the TCP connection to the peer fails.

The BGP-4 peer state machine utilizes timers. State transitions are triggered by information received from other IGPs, by BGP-4 packets received, by timers that expire, and by TCP connection state changes.

Chapter 5

Socket interface

Socket interface was originally developed for BSD UNIX. Socket interface offers high level abstraction of interface suitable to various network services. Socket interface was originally conceived because traditional “everything is a file” approach did not fit very well with network communications where operations such as “create connection” are necessary. Socket interface is based on the concept of a socket, which is basically a file descriptor with the additional property of being able to exist without association to a specific file. [25]

Socket interface is fully protocol independent. Socket interface is also very portable and all notable TCP stacks support it to some extent.

The socket operations that are discussed in detail on this thesis are introduced below. Only small subset of socket functions are introduced and specifications are summarily written. In addition a new interface is specified for creating sockets for protected TCP sessions. Excluding addition of `SOCK_PROTECTED_STREAM` service type, interface described here is identical to the regular BSD sockets.

5.1 Creating sockets

Call `socket` of the socket interface creates a new socket. The `socket` call takes three parameters: domain, type and protocol. A domain specifies the protocol family in which protocol belongs. As our product implements only IPv4 and IPv6, the domain is either `PF_INET` or `PF_INET6`.

Parameter type specifies the type of the service requested. Available service types include reliable stream, unreliable datagram service and reliable datagram service. Not all of these services are always available. For example IP protocol family has only reliable stream service (`SOCK_STREAM`) and unreliable datagram service (`SOCK_DGRAM`). Protocol type exist in this interface, because it enables an application to select a protocol with suitable characteristics without detailed knowledge of protocol family.

The protocol field allows an application to specify the exact protocol to use within the family. Usually this parameter is left to zero, and the default protocol for the requested service is used.

To facilitate creation of protected sockets, a new service type is defined. A service type `SOCK_PROTECTED_STREAM` creates protected socket. The service that is offered is same as with `SOCK_STREAM`, but in addition the connection is protected. The same interface could also be used for other protocols.

Call `socket` returns a socket number, which for all practical purposes is a file descriptor.

Various parameters of a socket are configured with the functions `setsockopt`, `ioctl` and `fcntl`. These functions are the least established part of the socket interface.

5.2 Establishing connections

Call `connect` of the socket interface tries to establish connection to a remote peer. `connect` call is only necessary for connection based services, such as TCP. `connect` takes a socket identifier and a remote peer address as parameters. The details of the remote peer address are not relevant for this discussion, but it suffices to say that address is in protocol independent form. Result of the connection attempt is returned. By default `connect` function blocks until connection is established or error is received.

A call `listen` is used to inform the protocol stack that an application is willing to respond to active connection attempts. A connection is accepted with call `accept`. These two calls are used to accomplish passive TCP open. Semantically `accept` for socket listening connections is identical to read from connected socket. `accept` call creates a new socket, and the original listening socket continues at the state `listen`.

5.3 Receiving and transmitting

Calls `send` and `recv` are used to transmit and receive data. A special version of these calls, which allows destination to be specified, exists for connectionless services.

Call `recv` blocks until there is the specified amount of data to be received, or a special condition occurs. Blocking of `send` call is subject to flow control.

5.4 Asynchronous I/O

Socket calls support asynchronous I/O. If asynchronous IO is configured for a socket with `ioctl` function, all of socket functions handling it return immediately. Status of socket operation can be polled with `getsockopt`.

As polling is not very efficient, the socket interface offers two additional choices: signals and `select`. A signal is basically a call-back function that gets called pre-emptively when operation has finished. Because usually real-time operating systems do not implement UNIX signals, this facility is not available.

`Select` calls allow a thread to wait for different kinds of events from several sockets. As `select` also supports timeouts, it is no surprise that `select` call is at the center of event loop in most routing daemons. `Select` can be used for all socket events, including for waiting new connections and errors.

5.5 Closing connections

Network connection is normally terminated by closing the socket associated with it. Actual behavior of termination depends much of protocol below socket layer. In connectionless protocols `close` just closes interface between the application and the stack, whereas with connection oriented protocols `close` call usually initiates closing of the connection. With TCP connection `close` call closes transmit side of the connection, but data can still be received (data will not reach the application, as socket descriptor is not valid after `close`). TCP connection can be terminated more selectively with method `shutdown`, which allows also termination of both directions of TCP connection. When a socket is closed, socket descriptor is released, and the descriptor then be re-used by the stack.

5.6 Inheritance of sockets

Sockets are always associated with tasks. When a new task is created, all open sockets of the parent process will be inherited by the new task. Due to peculiarities of the RTOS used, proprietary call exists that allows unrelated tasks to duplicate sockets to each other.

When two tasks share a socket, there actually is only one socket and two pointers to it. Both tasks may perform socket operations. A common arrangement is that one task performs reads the socket and another sends to it. A reference to the underlying socket structure is located on file descriptor table of each task that has the socket open.

Chapter 6

Current solutions for high availability routing

IP protocol family was originally designed for networks that could withstand a full scale nuclear strike. IP protocol tolerates errors very well: in best-effort IP networks' nodes contain no state and it is acceptable to lose some packets here and there.

Although IP network is stateless from the protocol perspective, all routers have much state information in the form of routing tables.

If IP network is built in redundant topology, such as ring or mesh, router or link failures do not prevent operation of the rest of the network. There is a small catch: to circumvent a fault, routers must notice it and reroute packets around it. [13]

The next five sections introduce various current practices for achieving fault tolerant IP networks. As shown below, they provide sufficient resilience for best-effort IP service. The technologies below are incapable of offering real carrier class resilience associated with traditional telecom networks.

6.1 Routing protocols

Dynamic routing protocols, such as RIP, OSPF and BGP-4 are used for distributing routing information on IP networks. All of these protocols have a way of noticing inoperative links and routers. RIP requires routes to be refreshed constantly. OSPF and BGP-4 use special messages to verify connectivity to neighboring nodes. When a faulty link or node is detected, the next best route is used – as soon as information of the broken link has reached other nodes.

By default OSPF will consider neighbor downed, if no hello is detected within 40 seconds. Convergence is then reached almost instantaneously. BGP-4 may take minutes to detect an error, but convergence is almost instantaneous. RIP might take anything from 30 seconds to several hours to converge. During the time between occurrence of a fault and convergence of routing information, not all of the routers in the network share same impression of the network topology. During that time routing loops and dead ends may occur in some parts of the network. [20]

An IP network running a non-RIP routing protocol recovers relatively soon from link and node errors, assuming network topology is sufficiently redundant. Many vendors, such as Cisco, offer also routers that can use dial-on-demand links to replace failed connectivity. Such arrangements can be used to reduce need for redundancy on small networks.

As regular IP networks recover from errors within a few minutes, they offer cheap and relatively robust solution for non-critical data transfer. For most purposes a few minutes of outage is acceptable. Sufficiently redundant network topology, and proper selection of routing protocols is all that is needed to guarantee this level of service. [13]

6.2 HSRP

The TCP/IP protocol family is based on the assumption that hosts are simple and all intelligence regarding routing is on the routers. In practice this philosophy means that most hosts have a default router that handles forwarding of the packets from the local network. Although this architecture reduces management overhead, it causes a single point of failure. If the default router fails, hosts within the router's network are unable to communicate with the rest of the world.

Cisco Hot Standby Router Protocol, HSRP solves this problem by creating cluster of default routers. When one of the routers fails, another will step in and take care of the tasks that belonged to the failed router. This solution elegantly preserves original paradigm of dumb host while removing the single point of failure. [17]

Each network has one or more routers that belong to one or more HSRP groups. Each HSRP group forms a single virtual router. The virtual router has an unique IP address and a virtual network level address. The HSRP protocol allows routers to vote an active and a standby router. The selection of the routers is based on pre-configured priority. After election only these two routers transmit HELLO packets while others remain silent. This behavior saves network bandwidth.

When the active router fails, it stops sending HELLO packets. The standby router assumes the duties of the active router, if no HELLO message is received within the pre-configured interval. Usually this interval is three seconds. When the standby router assumes the role of the active router, a new standby router is selected.

HSRP can also be used to protect upstream links – if the active router notices loss of upstream links, it can relinquish its position to the standby router which still has working uplink connections.

Although HSRP is a working solution, it has severe disadvantages. Although three second recovery time is acceptable for WEB browsing, it is hardly acceptable for voice over IP traffic. HSRP is also insecure, as only simple password authentication is used. The key concepts of HSRP are patented by Cisco.

6.3 Virtual Router Redundancy Protocol

Virtual Router Redundancy Protocol (VRRP) is an IETF standard that is in purpose and in functionality similar to HSRP. Unlike HSRP, VRRP is an open standard. VRRP achieves same goals as HSRP, but does it in more secure and flexible manner.

Although VRRP and HSRP share most of the functionality, there are some important differences. Perhaps the most important difference is that VRRP supports strong authentication methods whereas HSRP does not. The strong authentication of VRRP uses AH header to authenticate peers whereas HSRP can only use plain text password.

Another interesting difference is that VRRP uses raw IP datagrams, while HSRP uses UDP datagrams. VRRP protocol is always used to protect a real IP address of an actual interface. As mentioned in previous chapter, HSRP can only be used with virtual IP address. This difference has a few practical consequences, but presents a significant difference in design principles.

A group of routers participating in the protection of a single interface of a single router is identified with virtual router id (VRID). One group protects only one interface identified with one IP address. One router can participate to several different protection groups. This arrangement enables two routers to protect each other. Such arrangement can be used to protect load balanced connection.

Although VRRP is a good solution for the targeted problem, it only solves the problem of the failing routers at network's edge. [15]

6.4 Redundancy

Although no manufacturer currently offers fully protected IP routing solutions, some manufacturers offer systems with partially redundant hardware.

Almost all manufacturers offer both redundant power supplies and redundant cooling systems for their high end routers. These components are the most trivial to protect, and are also most prone to errors as they contain moving parts and high voltages.

Routers intended to core networks, such as devices manufactured by Juniper Networks, often offer redundant forwarding plane. Devices with redundant forwarding plane are capable of continuing forwarding, even if some of the switching hardware breaks down. These components are also often hot swappable. Redundant forwarding plane is common, because it is fairly simple to implement – all packets are forwarded according to static routing table that is now and then updated by the control plane.

The most advanced routers contain also a hot swappable control card. When a control card is removed, routing protocol connections are disrupted, but in some designs the actual forwarding may still continue. These designs allow limited operation, but only as long as the old forwarding table is valid and no other router stops forwarding. Inevitably after a while other routers notice the lack of response from route daemons on the failed router, and no packets are any more forwarded to the failed router. Hot swappable control card enables fast, but distrubtive replacement of the control card. For

uninterrupted service redundant hot swappable control card is a must.

6.5 Resilient transport

A common way of ensuring connectivity in IP networks is to run IP over another more resilient network technology. IP packets are often run over SDH and ATM core networks. Both of these technologies offer highly mature solutions for resilience.

Although using resilient network technologies under IP solves the problem with link failures, it also introduces some other problems. Stacking protocols excessively reduces bandwidth, especially if technologies differ. For example ATM cell tax reduces suitability of ATM for transporting IP datagrams. This waste of bandwidth raises costs associated with the connections. [11]

Another problem is, that resilient networks must at some point interface with IP routers. Unless these routers are protected, there is still the possibility of a failure.

6.6 Summary

Currently used high availability routing solutions are severely lacking. Protocols such as HSRP and VRRP allow primitive clustering of routers. Some of the best router hardware is partially redundant. Resilient transfer networks, such as SDH rings enhance reliability of links between routers. These solutions can not protect control planes of routers within network, and therefore failures are handled by slowly converging routing protocols.

Chapter 7

Protection

7.1 Concept

Protection is a term that is used to refer to a particular type of high availability solutions that are based on an architecture where one or more additional components can be used almost instantly to take over the functionality of the failed component.

Protection is commonly used in telecommunication networks to provide high availability telecommunications services. Most of PDH, SDH and ATM devices provide at least some protection features.

Protection is often divided to equipment protection and to line protection. Equipment protection refers to the duplication of the critical hardware components on nodes, such as line cards, control cards and switch cards. Line protection on the other hand means that several paths are reserved for the traffic to single destination. Usually paths go geographically separate routes, so that no single localized event may cause total failure of communications.

Line protection can be implemented with several separate mechanisms, that vary in both cost and switchover speed.

The 1+1 protection means that the traffic is sent with two paths and receiving node selects the best of the two signals. The 1+1 protection offers extremely fast switchover times, but is also expensive, as double capacity is needed.

The 1:1 protection means that there are two alternative paths for the traffic. Only one is used at a time and when the connection is disrupted, the traffic is directed to the alternate path. This protection method allows the secondary path to be used for low priority traffic, such as best-effort IP, which is dropped when switchover occurs. Unlike in 1+1 protection, the originator of data makes switchover decision. The switchover decision is based on signaling traffic. Obviously the reliance to signaling traffic for the switchover decision makes 1:1 protection relatively slow. This method offers slower switchover times than 1+1 protection, but is much cheaper to use, as extra capacity can be utilized to some extent.

A variation of 1:1 protection called n:m protection exists. The n:m refers to situation where for n path there exists m alternative paths. The method can be used for higher redundancy ($n < m$) or for

more cost effective protection ($n > m$). Obviously in later case, if all primary paths fail, not all of the traffic can be transferred to the alternate paths. Usually telecommunication industry bases failure scenarios to probabilities, and therefore n:m is the preferred protection method.

7.2 Applications

The obvious application of protection is to guarantee communication capacities in the moment of unusual event, such as equipment or line failure. Line failure may easily occur, for example due to construction work. Equipment failures occur because of manufacturing errors, software errors, and environmental catastrophes. As communication networks consist of thousands of devices, it is very likely that some of the devices become inoperable now and then. If network is large, these probabilities add up and it is probable that there exist one or more active failures on the network. In properly designed network these failures have very minor effect to the service provided by the network.

The other common use for protection is software updates and other maintenance work. Operators update software configuration quite often. They also restructure and expand their networks. Although change is occurring often, it is hardly acceptable that traffic would be interrupted for long periods because of it. Protection enables making changes to lines and equipment without interruption, as long as at least single protection path for each destination remains operable.

The incentive for telecommunication operators to ensure high reliability comes from both business reasons and from legislation. Obviously companies that need reliable connectivity for mission critical applications are prepared to pay premium for the connections. As telecommunication is a cornerstone of the information society, many countries have legislations that require certain level of reliability for the telecommunication infrastructure. It seems that when the environment is most harsh, for example during a flood or fire, communication connections are needed most.

Although most of the telecommunication equipment support protection, it is not always, or even often used. Protection is somewhat expensive technology, and it is not used for everything. While most of trunk connections are worth protecting, it would be forbiddingly expensive to protect the last miles of the POTS lines.

7.3 Protection in traditional environment

Protection in traditional telecom technologies, such as PDH, SDH and ATM is relatively straightforward, as traffic follows in pre-determined paths and signaling protocols are relatively simple.

Equipment protection is usually implemented by placing several identical components, and discarding output from passive ones. For example a TDM node could have a system bus where all data is sent, and two switch cards take data from the bus, and perform the switching. Output from the other switch card is discarded, and output from the other is used. Similar arrangement can be accomplished

for control cards.

Line protection is even easier, as one only has to program additional paths to switching hardware.

Main complexity of protection with traditional telecommunication equipment is related to management: how to manage two protected cards in a meaningful way.

7.4 Protection in IP environment

The actual routing in IP networks is no harder to protect than with traditional technologies. As long as routing tables remain synchronized, forwarding can easily be replicated – the only things that affects where a packet is forwarded are forwarding table and the packet itself. The IP protocol itself stores no state information in the network.

Protection of IP routing protocols is much harder to implement: there is a multitude of routing protocols to protect, and all of them have state. Most routing protocols have several inputs and timers, and it is not possible to guarantee, that by giving same inputs to two control cards they would produce the same output. As a direct result of this difficulty, there are no protected routers available from major vendors at the time of writing.

7.5 Benefits of protection

Protection offers unparalleled switchover times to any other available technology. Protection offers switchover times from almost zero offered by 1+1 protection to a few milliseconds offered by lesser protection technologies. Competitive methods based on routing procedures can offer times varying from seconds to several minutes.

Protection is in most cases the most cost efficient way to build high availability networks. Cost of acquiring, maintaining and managing redundant nodes would be immense. For non-critical networks protection is often unnecessary, as proper network topology planning and routing techniques offer sufficient level of service for much cheaper prices.

Chapter 8

Previous work on TCP protection

The chapters below review the previous work concerning the protection of applications using TCP as the transport protocol. All work reviewed here is oriented towards platforms where protected unit reboots and resumes operations. Methods based on this assumption are applicable, even if instead of reboot, switchover is performed.

8.1 Protocol extensions

One widely used method for solving the problem of protection for TCP based applications is to extend the application layer protocol to allow re-establishing TCP sessions. Such enhancements have been published for Label Distribution Protocol (LDP) and for Border Gateway Protocol 4 (BGP-4). In the chapters below these methods are presented in detail.

8.1.1 Graceful restart mechanism for BGP

A relatively recent IETF draft titled “Graceful Restart Mechanism for BGP” extends BGP-4 to allow re-establishing connections after control plane reboot. The draft is intended for architectures where actual forwarding continues while the control plane containing routing protocols is booted. [6]

Routers with this protocol extension can form peer relations with regular routers, but connection re-establishment is possible only if also the peer supports graceful restart extension.

The idea of the graceful restart is simple – while the control plane of a router reboots, the routing continues unaffected. When control plane is fully operational, BGP-4 connections are re-opened. The restarting BGP speaker waits until all the other routers have transferred all routes to the restarting speaker. The restarting BGP speaker will then replace its existing routes in the forwarding plane. The restarting speaker will then advertise its routes to other peers. The chapters below describe the graceful restart extension in more detail.

Graceful restart capability

A new BGP capability is defined to indicate support for graceful restart. A peer advertising graceful restart capability on the BGP open message agrees to generate an end-of-RIB marker. A BGP-4 speaker advertising graceful restart capability may also in addition of generating the end-of-RIB marker support forwarding during reboots for one or more address families. The capability advertisement contains also the following fields:

- Restart time
- Restart flags
- Flags for address families

The restart time indicates the maximum time that the restarting of the BGP-4 daemon may take in the worst case scenario. Usually this value is about one minute.

One of the bits in the restart flags field is defined as the restart state bit. The restart bit is used to indicate that the peer has restarted. Other bits in the restart flags field are currently undefined.

Each address family that supports forwarding during a reboot has separate flag that after a restart can be used to indicate whether or not forwarding actually took place during the restart. The flag is called as forwarding state bit.

Graceful restart capability is negotiated per connection basis. There is no requirement that all of connections of a router should support graceful restart – supporting and non-supporting connections can be mixed freely.

End-of-RIB marker

A BGP-4 speaker uses the end-of-RIB marker to indicate that it has no more routes to advertise. Justification for the existence of the end-of-RIB marker is that it allows a BGP-4 speaker to indicate that the initial routing update is over – that is all routes have converged.

For the traditional IPv4 address family the end-of-RIB comes in the form of minimal length update message: update message with no new route and no withdrawn routes. For other address families marker is an update message with MP_UNREACH_NLRI path attribute of that family with no withdrawn routes.

Initial connection establishment

When a BGP-4 speaker supporting graceful restart starts normally, it will try to establish connections to one or more pre-configured peers. When the BGP-4 speaker sends open message, it will advertise graceful restart capability. The restart state bit on the graceful restart capability advertisement is set to 1 to indicate that the router has no previous routing information. Forwarding state bits for all address families are set to zero to indicate that no forwarding took place during the boot.

If the other BGP-4 speaker does not advertise graceful restart on the open message it sends, neither BGP-4 speaker will use it.

The peering routers proceed to exchange route databases in the regular manner. When a BGP-4 speaker has sent all routes it is going to advertise, it sends the end-of-RIB marker to indicate it. When an end-of-RIB marker is received by a BGP-4 speaker, route selection is performed, and selected routes are advertised to other peers. The use of the end-of-RIB enhances BGP-4 performance, as no additional intermediate route selections or additional advertisements are sent.

Unless a crash occurs, rest of the session is identical to regular non-extended BGP-4.

Reboot of BGP-4 speaker

When one of BGP-4 speakers reboots, it may continue forwarding of packets for some address families. When the restarted BGP-4 speaker has booted, it marks all of its routes on the forwarding plane as stale. The BGP-4 speaker then tries to re-establish its peer relationships.

The receiving speaker (remote peer) may notice reboot based on error information received from the TCP stack. When it notices the loss of the connection, it will mark all routes received from the restarting speaker as stale. Normal BGP-4 speaker would in similar situation withdraw all routes originated by the restarting speaker. Marking a route as stale has no effect on the forwarding. A timer is primed with a period of the previously advertised restart time.

If the receiving BGP-4 speaker has not noticed the loss of the TCP connection, it will act as if previous connection was lost, when a new TCP connection is opened by the restarting speaker. An obvious denial of service attack, dropping legitimate routing connections by opening bogus TCP sessions, is avoided by using the optional MD5 authentication in all TCP segments.

When the restarted BGP-4 speaker sends an open message, it will set the restart state bit as 1. The receiving BGP-4 speaker will set the restart state bit in its open message as 0, as it has not restarted. The restarted speaker will only set forwarding state bit for those address families that were really forwarded during the restart. The receiving speaker withdraws all stale routes for address families which were not forwarded during the restart.

BGP-4 sessions to peers that do not support graceful restart are established according to traditional BGP-4 mechanism.

Re-establishing route databases

All remote peers of the restarted speaker advertise their routes to the restarted speaker. The restarting peer waits until it has received end-of-RIB markers from all its peers that support graceful restart.

When the end-of-RIB markers are received, the restarting speaker performs the route selection. New routes are installed to the forwarding plane, and all routes marked as stale are removed. If end-of-RIB marker is not received within reasonable time from all connections, the restarting speaker may perform route selection without receiving some of the expected end-of-RIB markers.

The restarting speaker proceeds to advertise Adj-RIBs-Out to all of its peers. It then sends the end-of-RIB marker. The remote peers will now perform route selection and install refreshed routes to the forwarding plane. Remaining stale routes are withdrawn. The restarting speaker has now fully recovered from the restart.

If IBGP is not used as IGP within AS of the restarting speaker, it is sensible for the restarting speaker to delay the route selection until IGP has converged.

Multiple reboots

Graceful recovery can only be used when BGP-4 speakers are initially in full operation. If the restarting speaker reboots during re-establishment of the session, the receiving speaker withdraws all routes originated by it. Dropping is performed with a simple rule: all stale routes originated by restarting speaker are withdraw, when connection is re-established.

Both speakers reboot

It is possible for two peering BGP-4 speakers to reboot simultaneously. On such occasion both speakers will set restart bit to indicate restart that has happened. The presence of restart bit prevents a deadlock, where both speakers would wait for the other to start.

This situations does not always prevent recovery, as it is unlikely that rest of the BGP-4 peers would have rebooted.

Timeout

If the restarting speaker does not re-establish a BGP-4 session to the receiving speaker, all routes originated by the restarting speaker will be withdrawn by the receiving speaker. The session between restarted and receiving speaker can be re-established later, but traffic disruption has already occurred.

Connection terminated by notification

If BGP-4 speaker sends notification message at any point of session, it must be handled with normal BGP-4 procedures. Graceful recovery is not possible for sessions that have been terminated with a notification message.

8.1.2 Fault tolerant LDP

Label Distribution Protocol, LDP, is roughly the same thing for multiprotocol label switching (MPLS), what BGP-4 is for IPv4 routing. LDP is used to distribute path information between MPLS nodes. Like BGP-4, LDP uses TCP as transport protocol. Also rules regarding lost connections are the same – if connection to MPLS switch is lost, all labels received from it are withdrawn. As MPLS is designed for core networks, fault tolerance is of premium importance.

Modifications to LDP protocol are almost identical to those introduced by the graceful restart mechanism for BGP-4. Fault tolerancy extensions for LDP introduce the following additions to regular LDP:

- An option to negotiate fault tolerant connections in backward compatible manner.
- A rule: if a LDP session has been negotiated as fault tolerant, labels transported over it are kept in use even if the underlying TCP connection fails.
- A rule: if TCP session is not re-established within short period with restore option, LSPs formed with the connection are torn down.
- A mechanism to replay messages that were not send because of TCP connection failure.

Although graceful restart for BGP-4 seems similar to fault tolerancy enhancements for LDP, there is also an important difference – LDP has mechanism for replaying lost messages, whereas graceful restart for BGP-4 is based on rebuilding route databases. Reason for different approaches is obvious: LDP has strong ATM roots and is mostly a signaling protocol, whereas BGP-4 is a routing protocol. LDP is much more dynamic in nature than BGP-4.

Both approaches would be applicable for BGP-4. The approach LDP takes is faster to converge, but also requires that control plane state information is not lost. The slower BGP-4 approach only requires that the state of the forwarding plane is retained. In this respect LDP approach would be more suitable for redundant BGP-4 routers. [8]

8.2 Wrapping TCP stack

A new idea to protect TCP stack was presented in IEEE Infocom 2001. The idea is based on wrapping TCP stack with all sides and using these wrappers to perform switchovers by tricking the TCP stack to the correct state. The method does not require any modifications to the TCP stack, as all protection functionality is in the logger and wrappers .[1]

The Infocom paper presents a simplified model for one application with a TCP stack that only supports one connection. The method presented in the paper is described below.

The Infocom paper makes the assumption that the application over the TCP stack behaves deterministically – any execution history must be repeatable by restarting the application and ensuring that `read` calls issued by the application produce identical results with the original execution. It is assumed that either switchover is same thing as booting, or backup card is connected to the same link as the primary card.

The basic idea of this method is to use wrappers to log socket calls issued by the application and data sent by the remote peer. During the possible switchover wrappers are used to fabricate packets that will in practice replay the TCP session preceding switchover. The replay is done both at the socket interface and at the TCP-IP interface.

Wrapping method introduces three new components to the protocol stack: north side wrap, south side wrap and logger. The north side wrap monitors and modifies socket call between applications and TCP stack. The south side wrap monitors and modifies communications between the TCP stack and the IP stack. The logger is located on a separate card. Figure 8.1 shows how these components fit in to the system.

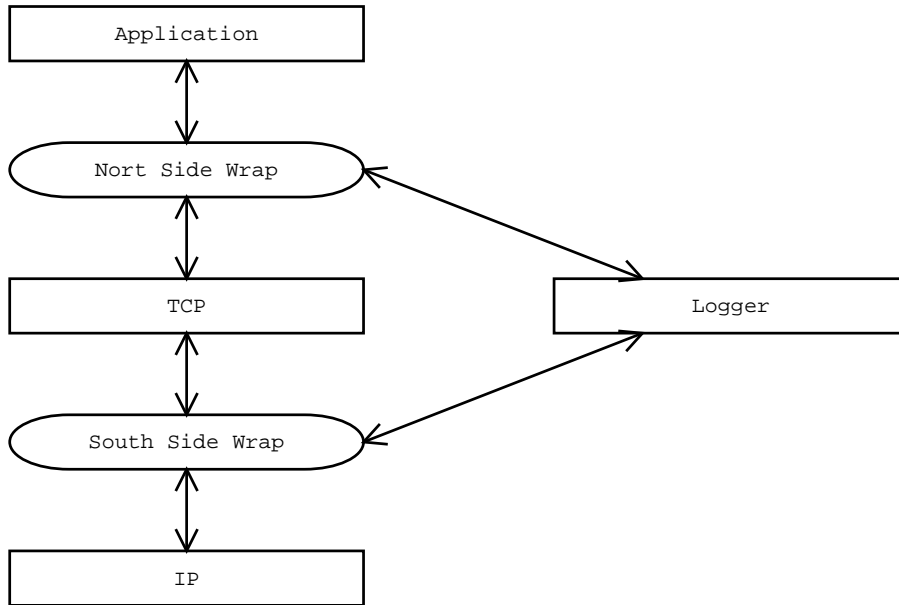


Figure 8.1: Wrapping TCP stack

8.2.1 North side wrap

North side wrap monitors and modifies socket operations that the application issues towards the TCP stack. The north side wrap is located between the protected application and the TCP stack. Responsibilities of the south side wrap are:

- Send lengths of all read operations to the logger.
- Ensure that replayed read lengths are identical to original ones.
- Discard all sent data during switchover.

Read lengths refer to length of data returned by calls to socket interface function `read`. The north side wrap monitors only length and order of socket operations – it does not interpret or modify actual data. Logging read lengths is performed asynchronously to guarantee the best possible performance.

8.2.2 South side wrap

South side wrap monitors and modifies traffic between the TCP stack and the IP stack. Responsibilities of the south side wrap are:

- Send copy of all received packets to the logger.
- Ensure that no unlogged data is acknowledged to the remote peer.
- Translate sequence numbers.
- Generate packets towards TCP stack during the switchover.

The south side wrap sends data to be logged to the logger. The south side wrapper also receives information about the amount of logged data from the logger. To reduce delays, logging of received data is asynchronous. When necessary the south side wrap rewrites, discards and generates TCP packets.

8.2.3 Logger

Logger is located on separate card or processor, so that failure in the protected component does not destroy logged information. The logger is responsible for logging received data and read lengths. The logger is also responsible for managing TCP connection during the switchover.

The logger is connected with both the south side wrap and the north side wrap with fast network connection.

8.2.4 Normal operation

When the TCP connection (only one connection is allowed by the model presented by the paper) is opened, the south side wrap logs initial sequence numbers to the logger. The south delays outgoing acknowledgment until the logger has logged initial sequence numbers. The north side wrap only passes through the connect or listen calls.

When packet is received from the IP stack, the south side wrap forwards it to the logger. To prevent delays, the south side wrapper also immediately sends the packet to the TCP stack.

When an acknowledgment is received from the TCP stack, the south side wrap rewrites acknowledgment number so that only data that the logger has acknowledged as logged will be acknowledged to the remote peer. Although this behavior enables safe asynchronous logging, it prevents efficient operation of TCP flow control. The problem manifests, because the remote peer believes that the bytes not acknowledged by the south side wrap are consuming TCP stack buffer and therefore must be reduced from send window. The flow control problem is avoided by rewriting window advertisement on the acknowledgment to include as many bytes of extra as there is data unacknowledged by the logger. Obviously IP checksum needs to be recalculated.

The north side wrap logs length of all read operations. Also this logging is done asynchronously. Write operations are only allowed when the logger has acknowledged all write operations that had preceded the read operation. Write operations are blocked until they are allowable. Ensuring log consistency before writes is important, as writes can also affect log content from the south side wrap.

Logger only receives read lengths from the north side wrap and received packets from the south side wrap. Received messages are immediately acknowledged to their senders.

8.2.5 Switchover

When logger notices switchover, it will assume responsibility of the protected TCP connection. The logger will send an acknowledgment with zero window advertisement as a response to all incoming packets. The acknowledgment only acknowledges received data up to the point it has logged. Sending zero window acknowledgments ensures that no further data has to be handled.

The control of TCP session is relinquished to the south side wrap of the protecting card (in the original paper to the south side wrap of rebooted card). The south side wrap continues generation of zero window acknowledgments.

The application performs either `accept` or `connect` call to establish the lost connection. In case of the `accept` the south side wrapper generates TCP segment containing SYN bit. Initial sequence number is same as sequence number of the last successfully received TCP segment. The segment is then dispatched to the TCP stack. Response to the SYN is dropped. Necessary acknowledgment is generated for the SYN. If the application performed `connect` call, the TCP stack sends TCP segment that has SYN bit set. The south side wrap drops the segments and generates proper response. The acknowledgment from the TCP stack is dropped. In both cases the initial sequence number of connection TCP stack was tricked to open is saved.

The TCP stack now believes it has open connection. Although the sequence numbers are correct for incoming traffic, they are not correct for the outgoing traffic.

The application will now go through the same sequence it did when it first started. The north side wrap ensures that both length and data of `read` operations are identical to the original execution. The north side wrap also catches all writes, and discards them. Discarding writes is stopped when the application has written as many bytes as the protected one did. The original amount of data written can be calculated from logged initial sequence numbers and sequence numbers of last sent packet before the switchover. When all data has been read from the log and all previously executed writes has been performed, TCP stack and application are synchronized. The north side wrap stops discarding writes and assumes regular operation.

The south side wrap stops generating acknowledgments zero window advertisements. As outgoing sequence numbers differ from the original session, translation is needed. The south side wrap performs this translation to sequence numbers of outgoing packets and to acknowledgment numbers of segments that are forwarded to the TCP stack. With this exception the south side wrap assumes normal operating mode.

8.2.6 Performance

The Infocom paper presented interesting benchmarking results. The benchmarks were performed with 450MHz Pentium II computer with 10 Mbps ethernet connection to remote peer and 100 Mbps ether-

net connection to the logger. The benchmark used an application that bulk transferred data from remote peer to the fault tolerant peer. Depending of the chosen acknowledgment strategy, fault tolerant implementation had throughput of 92-99 percent of the original TCP implementation. Benchmarks also showed that the bandwidth between the fault tolerant TCP stack and the logger is of major importance. With 10 Mbps connection between the fault tolerant TCP stack and the logger, performance was only 23-72 percent of original implementation.

The measured average switchover time for connection that had transferred one megabyte data was 22 ms.

It can be concluded that the method offers more than adequate performance for regular operation. The most important factor for performance appears to be the link speed between the wrappers and the logger. The switchover timer appears to be long because of time consuming log replay. This method is unlikely to scale well for long lived TCP sessions, as both log space requirement and log replay time increase with long lived sessions.

8.2.7 Requirements for applications

The wrapping method imposes the requirement that any application that uses the fault tolerant TCP stack must always behave deterministically when given same inputs from socket interface. If this is not the case, the recovery will fail, as the method relies on this behavior. For example BGP-4 is not deterministic in such way, as behavior not only depends from inputs from sockets, but also from content of the route database, timing between received messages and timing of messages received from IGP protocols.

The wrapping method is also unable to handle TCP applications that act both as a server and a client – not enough information is logged. As presented earlier, BGP-4 daemons behave both as clients and servers.

Due to these requirements for the application BGP-4 routing protocol daemons can not be protected with this method. This method would be suitable for protecting HTTP servers that serve static content. Anything more complicated than that would violate one of the above requirements.

8.2.8 Disadvantages

In addition to the unreasonable requirements imposed for the applications, the wrapping method has also some significant flaws. Perhaps the most inconvenient flaw is that logger has to store all data received on the protected TCP connection. While it might be acceptable for a short lived TCP session, it is definitely not acceptable for long-lived routing connection. It is entirely possible that protected BGP-4 connection could last years, if not decades and could transport gigabits of data. The sheer size of the log would prevent both reasonable storage and replay in any acceptable amount of time.

The wrapping method is unable to handle now widely used selective acknowledgment option. Supporting selective acknowledgment would in worst case require queuing packets on south side

wrap, as mere modification of acknowledgment number is not enough to ensure that logger has logged all incoming data.

The other problem related to acknowledgment handling is that the south side wrapper in practice delays acknowledgments until data the acknowledgment refers has been logged: when the acknowledgment number is rewritten by the south side wrap, acknowledgment for unlogged data is generated only when the TCP stack next time sends a segment. This delay manifests itself as incorrect round trip estimates by the remote TCP peer, and therefore leads to poor performance. The problem can be mitigated by introducing additional acknowledgments: the south side wrap generates additional acknowledgments when data has been logged. Several optimizations for this strategy is presented, but all of them increase round trip time estimate and use extra bandwidth for acknowledgments.

8.3 Transparent replication

A method where gateway or router replicates TCP session packets for both primary and backup has been suggested in a recent paper titled “Transparent TCP(IP based Replication” [9]. In the method the primary TCP stack executes as it would without protection. The backup TCP stack on separate card receives duplicates of all packets sent and received by the primary TCP stack.

The backup TCP stack uses leader-follower protocol to synchronise itself with primary: when there are several possible state changes for some input, the state transition is deduced by packet sent by the primary.

Perhaps the most critical weakness of this method is that it creates a single point of failure: the replicator between other routers and control cards. Also determining correct state machine transitions on the backup TCP connection state machine is hard and requires huge instrumentation of TCP stack. Due to these failures this method is unsuitable for the carrier class router platform, and will not be discussed more. A method similar to this one, but more suitable for carrier class router platform will be later introduced on this thesis. [9]

8.4 Summary

Protocol extensions for BGP-4 and for LDP represent elegant lightweight solutions. Both presented protocol extensions are less than six months old, and are not currently in general use. These extensions avoid complex TCP connection switchover with a simple modification to the protocol behavior. Wrapping the stack is impractical and is suitable only as an academic idea. Transparent replication is workable method, but it has a single point of failure.

Chapter 9

Requirements

9.1 Goals

Protection is used on the carrier class router platform to make it certain that a predefined level of services is available regardless of failures and ongoing maintenance. Following requirements are set for choosing the protection method:

- Minimal interruption to the traffic.
- Software version cross compatibility.
- Compatibility with third party routers.
- Suitability for other TCP based applications.
- Minimal changes to existing code.
- Minimal overhead.

The first requirement derives from the very definition of protection – uninterrupted service is the goal of the whole protection. The form of this requirement is purposely limited – a routing connection may be interrupted, if and only if it does not prevent forwarding. This requirement is the most important of the requirements. Obviously the lesser the interruption to the traffic is, the better the method is.

One of the important applications of the protection is updating the software running on the control cards. Regular routers, such as the ones made by Cisco, have to be booted when updated software is taken in use. With protection one of the control cards can be booted without interrupting the service. Therefore it is feasible to assume that there are situations where the active and the passive card have different software versions. This requirement is not absolute – in special occasions it might be acceptable that two software versions do not interoperate, as long as most of the software versions are compatible.

We must not expect to be the only vendor that provides IP networks. The protection method has to therefore interoperate with other routers. In practice this means that the protection is totally transparent to other routers. On the other hand it means that if there is a standard that helps other routers to offer better service, then it must be supported.

Although this thesis investigates protection only from the perspective of protecting the BGP-4 routing protocol, it is probable that a need arises later to protect other TCP based protocols. Therefore the method should be generic and applicable to other applications that use TCP as transport protocol. This requirement is not absolute, and if TCP protection can be avoided for BGP-4, there is no need to disqualify a good method because it does not support other applications. On the other hand if there are two equally good methods, the more generic of them should be chosen.

Developing routing daemons and IP stack represent a serious investment of manpower. Therefore some of the software components are bought or subcontracted. To ease re-integration of software to these modules, changes should be kept as minimal as possible. Minimally intrusive methods should therefore be favored over other protection methods.

As thousands of concurrent protected BGP-4 connections are possible, it is important that only little of extra overhead is caused by the protection method. The overhead includes CPU usage on the both cards, extra memory usage and communication bandwidth usage.

In addition to the technical requirements above, there exists a minor political requirement: the chosen protection method should not be patented. If the method is patented, a license must be obtainable with fair and reasonable terms. This aspect is not discussed further in this thesis.

9.2 Protected information

This section describes what data must be protected in order to achieve the goals described in the previous section. This section does not specify the exact technical implementation – although presented data items must be protected, protection method can achieve the protection by using copying, independent calculation or even re-obtain protected data from some other entity. This section does not specify any requirements.

9.2.1 TCP state information

The state of TCP stack has of two main components: socket structures and TCP connection states. Socket structures contain actual buffered data and information of pending calls. This data must always be protected or re-obtained by some means after switchover. Re-obtaining is only feasible if protection method allows re-opening TCP connection and losing all buffered data.

Minimum set of TCP connection state variables to protect contains negotiated MSS, current sequence numbers to both directions, send window, receive window, and TCP related timers. The MSS (maximum segment size) has to be known for proper operation of the TCP flow control. The sequence numbers must be correct or otherwise the remote peer will not accept further segments. Send and re-

ceive windows are necessary for smooth operation, although window size information is not vital. Timers need to be replicated to facilitate proper retransmission.

Obviously also the indication of the current connection state should be protected for each active connection. There is no point of protecting TCP connections that are not in the established state – BGP-4 does not utilize half-duplex states (states FIN-WAIT-1 and CLOSING), and half-open state can always be re-negotiated without lessening service level.

9.2.2 BGP-4 state information

The first of the three route information bases of BGP-4 (Adj-RIBs-In, LocRIB and Adj-RIBs-Out) must be protected. Two others can be re-constructed with expensive calculation from the first route database and from the system route database. As this calculation is very expensive, in practice all three RIBs should be protected. It should be noted that RIBs are logical structure defined to facilitate understanding of BGP-4 protocol, and therefore an actual implementation may differ from this model.

State information of peer state machines must be replicated. This state contains protocol state machine state, active timers and various negotiated options and capabilities.

9.2.3 Other protected information

Routing protocol filters, policies and configuration have to be protected. Also system route database must be protected. Protection of these elements is handled by system software. This thesis makes the assumption that these components are always up-to-date on both cards.

9.2.4 Summary

Proposed solutions shall be compared with several well defined criteria. The most important criteria is minimal interruption to the traffic. Also business reasons, such as required changes to third party code are considered.

A sizable set of protected data can be derived from these requirements. The protected data contains TCP and BGP-4 state and configuration information.

Chapter 10

Bulk transfer methods

All protection methods need to transfer some data between the active and the passive card – at the minimum when performing initial synchronization. Some method might be otherwise independent, but others might keep copying state information continuously. There are two distinctly different ways of copying state information from the active card to the passive card: memory copying and encoding of data.

10.1 Memory copying

The memory copy method is based on the idea that all protected data is placed on a separate memory area that is copied to the passive card whenever bulk transfer needs to be performed. The relocation is relatively easy to perform, as on RTOS that is used, all modules have separately relocatable memory allocation pools.

There are several issues that need to be ensured when implementing this type of copying. First thing is that the whole state of memory allocator must be located on the memory area that is replicated. That is the case with our RTOS of choice. Another obvious issue is that synchronization must be performed to guarantee that copies are consistent and bulk transfer does not interrupt memory access operation.

The receiving end of the transfer must not receive the replicated memory area over the previous copy, as it is possible that the active card might fail in the middle of the bulk transfer. This presents no problem, as our hardware has MMU that can be used to re-map received memory area to its destination location when the bulk transfer has been successfully performed. The side effect of this method is that the protected partition of the memory consumes double the space on the passive card.

For those protection methods that perform more than one copy operation, there exist an intuitive optimization – MMU can be used to keep book of changed (dirty) pages, and further copies can be done incrementally.

On the carrier class router architecture it is possible to perform actual bulk transfer by using proprietary features of the backplane interface and programmable scatter&gather capable DMA unit.

This leaves CPU free and the only penalty is consumption of some of the bandwidth of the memory bus. It should be noted that there is bandwidth of several gigabits per second available for bulk transfer on the bus between cards, but there is not too much of extra capacity on the memory bus.

As slowest part of the chain from memory of the active card to the memory of the passive card is PCI bus interface to ASIC, it is easy to estimate that maximum transfer rate is 132 megabytes per second. Buffers on ASIC and on the memory controller reduce this estimate slightly. To account for other PCI bus traffic, we can reduce the bandwidth estimate to half. At this speed 100 MB worst case initial synchronization would take 1.52 s and a change of 32 kB would take 0.42 ms.

To guarantee software version compatibility, all data structures must be generalized so, that small changes are later possible without breaking compatibility to older software versions.

This method has several advantages. The most important of them is that modifications to underlying applications are minimal. This feature comes from the fact that this method does not understand data it is copying. Also this method is very fast and does not consume precious CPU time.

The most important disadvantage of this method is that only minor changes are possible between software versions. For example version 1 uses linked lists for something and version 2 uses arrays, there is no way that bulk transfer is possible between these versions. Other disadvantage is that if MMU optimization is not used for selecting only changed pages, copying might take a while and hinder memory accesses by the processor.

10.2 Encoding data

The second method for bulk transfer uses protocol specific encoding routines that traverse all necessary data structures and serialize protected data. Serialized data is then transferred to the passive card where data is deserialized and data structures are re-created. Re-created data-structures contain same information as in the passive card, but possibly in different form or containment.

Obviously this method produces minimal size set to copy with cost of additional processor consumption. The major disadvantage associated with this method is that encoders are needed for all structures present in protected data. Therefore this method requires heavy instrumentation of third party code.

Advantage of this method is that as the form of the state information is standardized, software compatibility between the active and the passive card is much easier to obtain.

10.3 Conclusions

Memory based method should be used, as it is superior in most aspects compared to the encoding based method. The decisive factor against the encoding based method is that heavy instrumentation of third party code is unacceptable.

The issue between software incompatibilities can be solved when it raises – conversion routines

can be written. As conversion routines would be executed on the passive card, load would be more evenly distributed, as most of the time the passive card is more lightly loaded than the active card.

Chapter 11

Protection methods

11.1 Method 1: logical synchronization of TCP and BGP

As both the active card and the passive card receive the same packets, it is possible to run TCP and BGP-4 state machines independently on both cards. The output related to protected TCP connections from the passive card is discarded until switchover.

If this approach is used, initial sequence numbers for new connection must be chosen identically in both cards. As random numbers are produced by a strong random number generator, identical sequence numbers must be accomplished with means of message passing.

11.1.1 The need for synchronization

As the active card has more tasks to perform than the passive card, it is likely that programs on the passive card have better response time. It is also likely that cards have different software versions in use. Executions on both cards are in no way synchronized. If state machines are executed independently on both cards, there is a high probability that their execution histories will diverge sooner or later. This divergence can be caused by several reasons.

The most likely reason for divergence is that acceptable acknowledgment ranges for some protected connection differ between cards. If TCP transmit operations are asynchronous, the first card to send the datagram has larger acceptable acknowledgment range than other. If the active card sends data first and acknowledgment is received from remote peer before the passive card has transmitted the segment referred by the acknowledgment, TCP state machines will diverge: the passive card will forever try to retransmit and the active card continues normally.

Divergence can also occur because of timers. If the link to a remote peer has been unavailable for a while and TCP stack on both cards are about to drop the connection, following events could occur: remote peer sends acknowledgment, the lightly loaded passive card processes acknowledgment immediately and keeps the connection, while on the highly loaded active card a timer expires before the acknowledgment is processed.

The timers on BGP-4 can also cause similar situation. Further on differences on BGP-4 state can easily lead TCP differences as routing daemons would try to send differing traffic over TCP connections.

11.1.2 Logical synchronization

One possibility is to synchronize both TCP and BGP-4 states. This is not feasible, as the cards do not have a common timing source, and it is likely that software versions differ. Another possibility is to perform synchronization logically: even though clock timing of events differ, their order is the same. It should be noted that synchronization (ordering) for TCP must be done on connection basis, as otherwise the required buffer space would be immense and performance impact might be huge.

Logical synchronization is accomplished for TCP stack with a simple arrangement: always when the active card is about to perform an operation on protected connection, it will send message to the passive card which will only then perform the same operation. This guarantees that operations are performed in the same order on both cards. The chapters below describe the exact behavior of each participating component. Picture 11.1 shows general overview of this method.

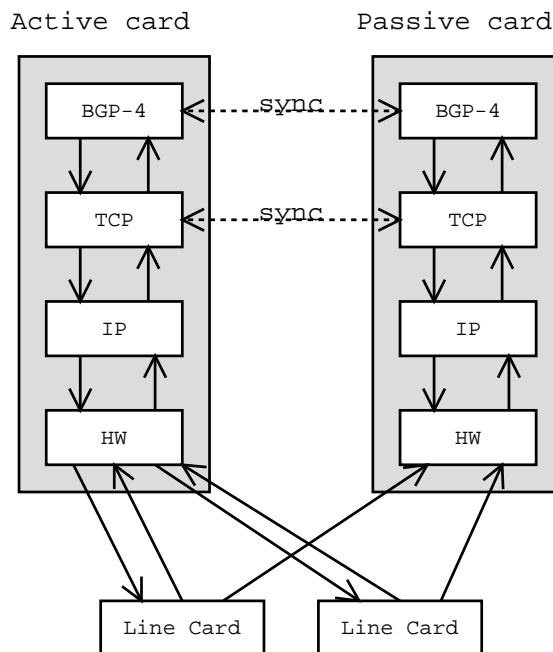


Figure 11.1: Overview of method 1

Protected connections can be identified on FreeBSD based stack easily – if a flag indicating protection status is added to TCP control block of the connection (TCB), the flag can be accessed both based on socket number and in `tcp_input` function. Identification and synchronization of the protected TCP segments must happen in `tcp_input` function immediately after the TCP control block is identified based on source and destination IP addresses and ports.

It should be noted that hash tables indexing both protected and non-protected data structures need to be split to prevent corruption during initial replication.

TCP synchronization

Following operations within TCP stack must be logically synchronized for each protected connection:

- Timers
- Transmit of TCP segment to network
- Receive of TCP segment from network

All operations for non-protected connections are asynchronous to avoid imposing unnecessary overhead.

Timers of a protected connection must be prevented from expiring unless they have expired on the active card. This can be accomplished by adding a flag to the timer structure, and expiring the timer only in two occasions:

- We are active card and it is time to expire the timer.
- When ITC message from active cards says that timer has expired.

As all previous events are ordered, there never is a situation where timer referred by expiration announcement does not yet exist in the passive card.

Transmit of TCP segment must be synchronized. It can easily be accomplished by queuing transmit operations until message is received from the active card. Transmit operations are defined to be sending some part of the socket buffer or sending an acknowledgment.

Also received TCP segments must be queued and processed only when the active card is starting to process them. Some sanity checks could be performed to drop obviously incorrect packets without need of synchronization.

The messages that activate operations must be in the form: (connection, operation, id), where connection is a unique connection identifier, operation identifies which type of event has been performed on the active card and id is unique identifier for the operation to be performed. Identifiers can be chosen sequentially, as synchronization guarantees that both cards will choose same identifiers. The operation described by synchronization message must be performed as soon as possible (immediately if it has already been issued) so that events occur in order.

Socket synchronization

Socket operations are conceptually separate from TCP stack, but share same many data structures. Socket operation ordering affects directly to synchronization of the TCP stack, and care must be used with defining their synchronization behavior to avoid paradoxes. Socket calls must be synchronized

because much of TCP behavior depends on the state of the socket buffers. Synchronization is shared between sockets and TCP, so in other words both socket and TCP operations are queued on the same queue and dequeued in same order as in the active card.

Most of the calls on socket interface may block. Unblocking of blocked socket calls requires no additional synchronization, as unblocking is direct result of state changes in synchronized TCP connection state machines.

Procedure `socket` is used on socket interface to create new sockets. A special version of socket procedure is provided for creating protected sockets. This procedure does not change TCP state machine, and is therefore not synchronized. To avoid socket descriptor collisions with non-protected sockets, different name-space must be used: large numbers can be used for protected sockets whereas socket handles for regular sockets are small integers. Some mechanism must be used to guarantee that same applications get same socket identifiers on both cards. Socket handle could, for example, be specified as the parameter of the procedure.

Procedure `close` is the opposite of procedure `socket` – it closes TCP connection and frees socket identifier. Only required modification for the `close` is the synchronization. Synchronization can be performed on per-connection basis.

Procedure `connect` is used for creating new TCP connections. As `connect` is associated with connection it must be synchronized with opening of the same connection on the passive card.

Procedure `send` is used for sending data to TCP stream. In our implementation there exist no separate `write` procedure for sockets – calls to `write` are mapped to `send`. Usual per-connection synchronization suffices.

`Recv` procedure must also be synchronized per connection basis. No separate `read` procedure is implemented, as used RTOS has no centralized file management. Calls to `read` are mapped to `recv`. `Recv` procedure must be synchronized, as the amount of free space on the socket buffer directly affects to TCP flow control.

Although procedure `select` changes nothing on the TCP state machine, it has to be synchronized to guarantee that all operations before it have been completed. If synchronization would not be performed, application on the active and passive cards could get different results with `selects` on multiple socket descriptors. This is especially vital, as BGP-4 daemon relies heavily on `select` procedure.

Asynchronous IO with signals is not required and therefore we avoid considering synchronizing it. The support functions `fcntl` and `ioctl` must be synchronized, as some of their parameters affect directly to TCP state machine.

BGP-4 synchronization

BGP-4 routing daemons can be synchronized with similar arrangement to TCP. The events to be synchronized are:

- BGP-4 hold and keep-alive timers

- Socket operations
- Routing events
- Network management actions

BGP-4 hold timers can cause termination of connection. These timers must be synchronized with the other control card to ensure that the state machines do not diverge. Keep-alive timers have to be synchronized, as management actions may terminate connection and expiring keep-alive timer could cause different send operations to be performed at the passive and the active cards. Same technique used for TCP timers applies also here.

Socket operations also have to be synchronized at the BGP-4 daemon. Although protected sockets will be synchronized at socket level, `recv` operations have to be synchronized to guarantee ordering with timers. Synchronizing write operations is not so vital, but should be done to guarantee that management operations won't cause strange situations within TCP/IP stack.

Routing events has to be synchronized to guarantee ordering of other events.

Network management actions that can affect ordering of other events must be synchronized. It is most likely easiest to synchronize all management actions.

11.1.3 Switchover

At the moment of switchover the passive card becomes active. As TCP state machines for protected connections are in correct state, the role of active card can easily be assumed. The only thing that TCP needs to do is to execute any events pending synchronization from the old active card.

The BGP-4 state machine must also to execute all events pending synchronization from the active card.

The switchover is instantaneous, within limits of hardware fail-over, except in a single special case. The special case is, if the active unit fails in the middle of transmit and the passive card is also at the middle of transmit. In this situation no segment gets sent even though the new active card believes it has sent one. This will not cause a longstanding problem, as TCP retransmit mechanism will correct the situation.

One might try to change situation by modifying synchronization strategy to such, that the passive card executes corresponding event only after the active card has finished the event. Although such strategy would be possible, a more complex definition for blocking socket operations would be necessary – socket operations can cause other events in TCP stack. For example socket operation `send` can cause transmission of TCP segment, an operation that can finish before the blocking `send` call does. Result of this is that passive card is instructed first to perform the TCP segment transmit operation and only after that the socket `send` operation. This combination is often impossible, as when the transmit buffers are empty, socket `send` operation must occur before TCP segment transmission.

11.1.4 Insertion of new control card

There are four scenarios for control card insertion/boot:

- Node is booted with only one control card.
- Node already has an active card
- Node is booted with two control cards
- Node that has no control cards.

In first scenario, no replication is necessary for successful boot. As hardware automatically selects active card when two control cards are present, third scenario practically is the same as second one. Both of these scenarios require initial replication. If node has no control cards, no action is necessary, as node is shut down when the last control card is taken off-line.

When a new passive control card is inserted, both the BGP-4 routing daemon and protected TCP state machines have to be initialized in that card. One of the bulk copy methods presented earlier could be used to replicate necessary data structures. As no call stacks are replicated, the software on the active card must perform the replication in such state that all protected state information resides in replicated data structures.

It is important to guarantee that the BGP-4 routing daemon and and TCP state machines will remain synchronized, when both daemons are enabled. This can be accomplished with the following sequence:

1. Disable emitting any new (routing and timer) events on the active card. Drop all incoming TCP packets on both cards.
2. Perform synchronization.
3. Enable the TCP stack on the passive card.
4. Enable the BGP-4 stack on the passive card.
5. Enable the TCP stack on the active card.
6. Enable the BGP-4 stack on the active card.
7. Enable emitting events on both cards.
8. When first synchronization message of TCP receive event is received by the passive card, the passive card drops all TCP packets that have been queued before the packets which the synchronization message was for.

The route database must not emit any events during the synchronization. If route database would emit events and BGP daemons would just queue them, it is possible that the active card might have more events on the queue than the passive card.

TCP stack is enabled on the passive card before it is enabled on the active card. This arrangement guarantees that the passive card has at least all the data the active one has. The step 8 of above sequence drops unnecessarily queued data. This somewhat complex arrangement is necessary, because there is no way to start TCP stacks at exactly the same moment on both cards.

11.1.5 Signaling diagrams

Insertion of control card

Figure 11.2 shows the signaling diagram of the insertion of new passive control card. In the diagram control entity presents system software within the cards.

When the passive card is booted, the card sends message to the active one and request synchronization. When the passive card is booted, both the protected TCP stack and BGP-4 stacks are suspended right after completing their initialisations. The active card suspends generation of new events from the route database. This must be done to guarantee that both BGP-4 state machines will get same events after synchronization is complete. Then both BGP-4 and TCP stacks are suspended. BGP-4 stack is suspended first to prevent additional data to be written to socket buffers during the synchronization.

BGP-4 and TCP stacks are synchronized using one of the methods presented earlier. Route database synchronization is handled by other subsystems but conceptually happens at the same time as TCP and BGP-4 synchronization.

After synchronization the active card advises the passive card to enable BGP-4 and TCP stacks. The passive card informs the active card when stacks are enabled. It is vital that the passive card enables TCP stack before the active card, so that its TCP receive queue contains at least all the packets that the active card has.

The active card then enables TCP and BGP-4 stacks. Events from route database can be enabled asynchronously, as they will be synchronized and BGP-4 stack.

Not drawn in the signaling diagram: when the active card processes next incoming TCP segment, the passive card will discard all TCP packets before it and will at that moment be synchronized with the active card.

Send and receive

Figure 11.3 shows a scenario where BGP-4 timer initiates sending of TCP packet (eg keep-alive) and soon after a packet from the same TCP connection is received.

Timer TIC message from operating system timer is received by both BGP-4 stacks. BGP-4 internal timers are evaluated on the active card and the passive card. As the passive card got tick message before the active card, it flags timers as queued. As timer x expires, synchronization message is sent

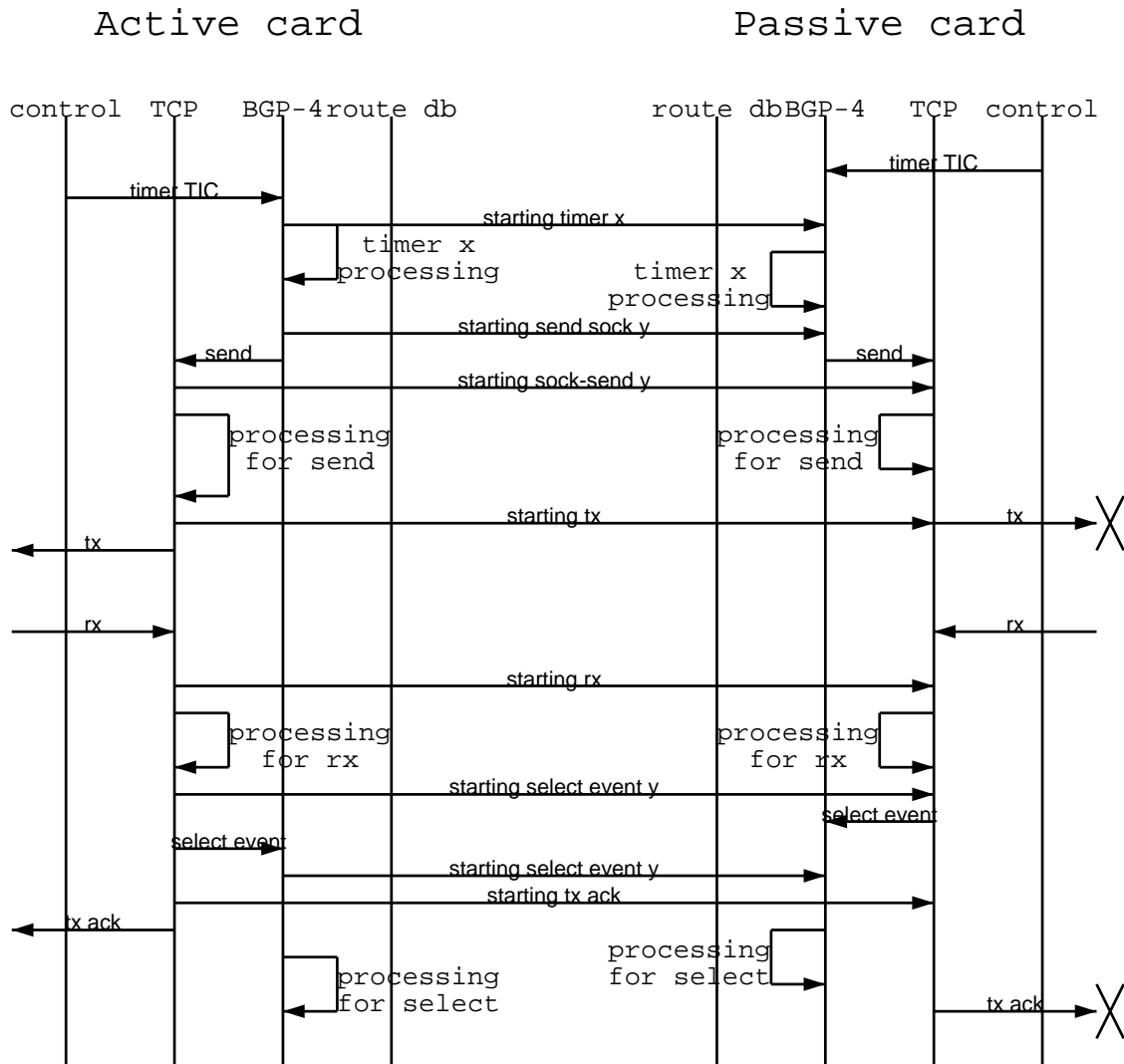


Figure 11.3: Send and receive (method 1)

card can process the event first, as scheduling might cause delays on the active card. The BGP-4 daemon will delay processing select event until the active card informs it is going to process it. As BGP-4 and TCP daemons are synchronized separately, it is possible and likely that events between them occur in different order, as is the case with sending ACK in TCP stack and processing select event on BGP-4 stack. Although events can occur in different order, ordering within protocol (TCP/BGP) is always the same.

Switchover

Figure 11.4 shows a scenario where the active card crashes in the middle of a BGP-4 timer execution. Similar scenario for TCP timer is analogous and is therefore not shown.

In the scenario presented by the diagram BGP-4 daemons in both cards receive timer TIC message

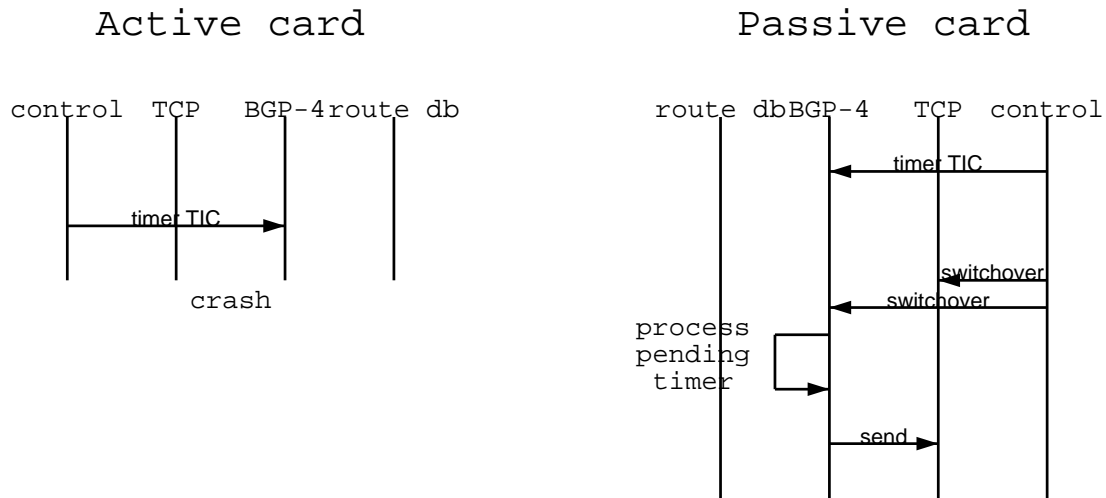


Figure 11.4: Switchover (method 1)

from the system timer. The passive card queues timer and waits for synchronization message. The active card crashes before it has a change to send the synchronization message to the timer. Crash is detected almost instantly by the hardware and BGP-4 and TCP stacks are notified of the switchover.

As the result of the switchover, BGP-4 daemon executes the pending timers. All operations that follow are asynchronous (until new passive card is inserted or original active card recovers). In this example timer processing caused send socket call. Rest of send call is not relevant for the protection and is not shown in the diagram.

In the figure 11.5 is shown a scenario where the active card is in the middle of TCP send operation and the passive card has already sent its packet. It is the only scenario where re-transmit has to be used to synchronize the state at switchover. It is also the only case where switchover is not nearly instantaneous.

As operations are synchronized before they are executed, passive card sends the TCP segment to be transmitted before the active card crashes. Output of the passive card is discarded. As the active card crashes, no segment gets sent. Some time after the switchover retransmit timer expires on the passive card. As switchover has happened, no synchronization is performed for the timer. The retransmit is performed according to normal TCP retransmit behavior. As the former passive card is now active, retransmit is not discarded by the hardware.

11.1.6 Advantages and disadvantages

This method is relatively easy to implement and maintain, as modifications are only required to relatively few event passing points. Required communication between the active and the passive card is minimal. The method seems to be very suitable for our hardware architecture, as it utilizes automatic packet replication service of incoming packets. Switchover is also very fast.

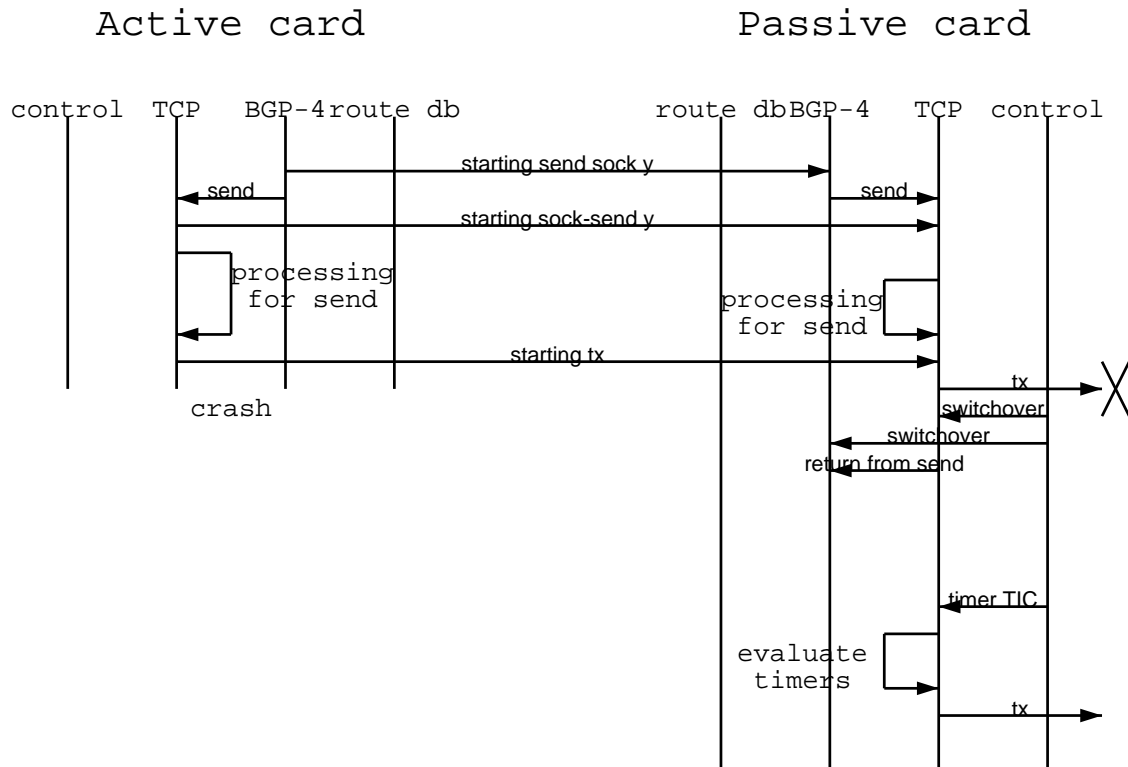


Figure 11.5: Switchover while sending(method 1)

Solution is generic and is not limited only to BGP-4. The catch is that protected applications must be aware of the protection – protected applications on both cards must perform same socket operations in the very same order. In practice this means that protected applications have to be synchronized. This requirement causes a serious limitation – behaviour of BGP-4 daemon must be identical on both cards. If software would be updated in a such way that socket operations would differ, upgrade utilizing protection would not be possible without special considerations. Protected application must not use any unprotected sockets, as otherwise switchover would fail.

To guarantee binding between task and its socket descriptors, this approach requires that tasks have same task identifiers on both cards. Task identifiers could change, if creation order of tasks changes. This requirement limits how much changes can be done to software while maintaining software update capability. This limitation can be avoided by using pre-defined task ids for protected tasks.

A disadvantage is that the initial synchronization might be complex. Some of the complexity of the initial synchronization can be avoided, if memory copying is used as the initial synchronization method. On the other hand no additional replication after initial synchronization is necessary. Constant synchronization also limits maximum possible performance.

11.2 Method 2: replicating TCP state

In this method, the TCP state machine is executed only on the active card. Relevant TCP connection and socket state structures are replicated to the passive card. The BGP-4 daemon is executed on both cards. Non-protected partition of the TCP stack is of course executed also on the passive card. Similarly to method 1, this method does not tolerate shared hash tables between protected and unprotected parts of the stack.

Replication has to be performed on such granularity, that if the active card fails, the passive card has enough information to continue TCP sessions. In practice this means that TCP connection and socket state structures have to be replicated:

- Before completion of most socket calls
- Before sending data
- Before sending acknowledgment

Call stacks and task scope variables are not replicated, and therefore replication points must be chosen so that all necessary information for maintaining TCP connections is on replicated dataset. In practice this means limiting replication points to event loops and carefully selected locations on socket interface.

As the protected TCP stack is not executed on the passive card, all the responses to socket operations for protected connections are received from the active card. In practice this means that most socket operations on the passive card are put to hold, and during the replication process a message from the active card tells what should be returned with what socket operation.

The replication of protected TCP connection state structures and socket structures can be performed with one of the bulk transfer methods presented earlier on this paper.

The BGP-4 daemon on the passive card must be synchronized with the active card to guarantee that socket operations are performed in the same order. BGP-4 daemon is synchronised with method presented on chapter 11.1.2, but other methods could also be used, as long as socket operations are guaranteed to be in the same order.

The picture 11.6 shows a general overview of this method.

If the passive card crashes, the TCP connection state machines on the passive card are either in correct states, or can be brought to the correct state by executing all pending socket operations or waiting for retransmissions from remote TCP peers.

11.2.1 Transmitting TCP segments

Sending TCP segments changes one critical element of TCP connection state, the acceptable acknowledgment range. Therefore replication has to be performed whenever TCP stack is about to send a new segment.

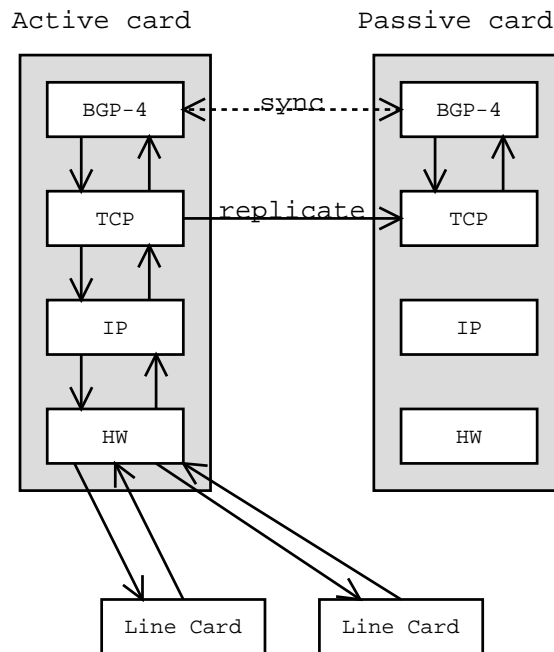


Figure 11.6: Overview of method 2

The replication has to be performed in such order, that the acceptable acknowledgment range includes data to be sent, and the data is sent only after replication. This order guarantees that if the active card crashes after transmit, the passive card is in the position to accept the acknowledgment.

Re-sending segments does not require replication, because the passive card may perform additional re-sends after switchover. The only harm caused by using re-transmit in this way is the minute amount of network bandwidth used during switchover.

11.2.2 Receiving TCP segments

There is no need to perform synchronization immediately when data is received – unacknowledged data will be resent by the remote peer, if the active card crashes, and therefore does not acknowledge the received TCP segment.

When the acknowledgment is sent, normal transmission synchronization presented in the previous section is used to replicate the new state to the passive card. Synchronization has to be performed before sending an acknowledgment, because when acknowledgment is sent, our node has committed to keep the data on local buffers, and there is no way we can re-receive it.

11.2.3 Socket interface

All operations for unprotected sockets are executed normally by the passive card. All operations for the protected sockets are queued on the passive card, and results received by messages from the active card are returned to the caller. The message must contain operation identifier, socket number, process

id of process where socket id is meaningful, and the return values of the operation. If socket operation referred by a message from the active card has not yet happened, the message is queued until the BGP-4 daemon on the passive card issues the same socket operation.

The socket operation `send` must perform the replication of the TCP connection state structures, as it adds data to transmit buffer. If the flow control does not block the call on the active card, replication is performed immediately after the call. As part of the replication a message is sent to the passive card, indicating the return value of the call. If the `send` call is blocked by the flow control, TCP connection state is replicated. During replication a message is sent to the passive card indicating that the `send` call blocks. When `send` call is able to return, another replication is performed and the active card sends message indicating the return value of the call. The passive card keeps log of the state of `send` call, so that if the active card crashes in the middle of a blocking socket operation, the passive card does not replay the whole socket operation.

There is an additional complication for `send` operation – often large segments are transmitted immediately without buffering. In such case a replication is performed before the TCP segment transmission. During this replication the passive card must flag the queued `send` call, so that in event of switchover only the part of `send` call following the transmission will be executed. Another possibility would be to eliminate possible double replication by forcing all transmitted data to be buffered. Option to synchronize only once at socket level would not work, as acceptable acknowledgment ranges would differ during the time interval of transmission and replication.

The `read` function performs replication, because it removes data from the receive buffers. The replication must be performed just before a `read` call would return, as it is the moment when data is removed from the buffers. At that moment also message indicating the return value is sent to the passive card. The message contains also the data that the `read` call will put to the memory region pointed by the caller. Call `read` may block when less than requested amount of data is available at the receive buffers. Blocking requires no special handling, as `read` calls require replication only upon completion of the call. It is possible to optimize `read` call so, that instead of the active card sending the data returned by the `read` call, only pointers to receive buffers are transmitted. Such optimization means that replication becomes more complex, as the replication would normally remove `read` data from socket buffers.

Socket function `select` almost always blocks. The `select` call does not modify state structures and therefore does not require synchronization. Instead `select` call is queued on the passive card. When the active card detects changes on sockets monitored by the `select` call, it sends the return value of `select` call and the correct bit masks of the file descriptors to the passive card. Then both the active and the passive card wake the calling process. Because unprotected sockets are not replicated, `select` call may only operate on purely protected or unprotected socket descriptor sets. Calling `select` for set containing both protected and unprotected socket would cause unpredictable behavior.

Call `socket` requires replication to guarantee that identical sockets are created on both sides.

The call `socket` never blocks and replication is always performed before return at the active card.

Call `listen` never blocks. Replication has to be done to update the passive card of changes in the TCP connection state machine. The return value of the call is sent before return by the active card.

Socket function `accept` blocks almost always. As the `accept` call is semantically similar to the read call, blocking behavior is identical – passive card queues the `accept` call until message from the active card is received. When the `accept` call would return, replication is performed to copy new socket to the passive card. Message is then sent by the active card indicating the return value of the `accept` call. Unless `accept` returned with error, this return value is the socket identifier of the newly created socket.

Function `connect` blocks until a connection has been established. The passive card queues `connect` call until the return value from the active card is received. As `connect` involves state changes to the TCP connection state machine, replication has to be performed before blocking the `connect` call. A message must also be sent to the passive card before the blocking, as after potential switchover the passive card must be able to distinguish between `connect` calls in process and `connect` calls that are waiting for response.

As most routing daemons only use asynchronous socket I/O, it would be feasible to limit the implementation to support only asynchronous behavior on the protected sockets. Such decision would greatly reduce complexity of the socket interface modifications, as the behavior for each socket interface call would always be the same: replication is performed, and the active card sends the return value in message to the passive card.

11.2.4 BGP-4 synchronization

The synchronization of BGP-4 events on this method is identical to one that is presented in chapter 11.1.2 for method 1. Also the reason for synchronising is the same.

11.2.5 Replication

One of the bulk copy methods presented earlier is used to transfer the state information. Probably the best method would be MMU assisted memory copying, as it would cause minimal overhead and almost no instrumentation of the TCP stack code.

During the copy operation the TCP stack on the active card must queue all incoming TCP packets and socket operations. Most likely copying can be performed so swiftly that it is acceptable to prevent scheduling during the replication and by that guarantee consistency of the replica.

If transmitting the return value is performed with the socket operation, the return value must be transmitted in the same operation – both the return value and the replication must succeed or neither may. This rule guarantees that restarted socket operations start from consistent state.

11.2.6 Switchover

In the trivial case the passive card has current state information for protected connections and TCP can take over as if nothing would had happened. There are four special tasks that usually have to be taken care of at the moment of the switchover:

- Interrupted re-sends must be handled
- Unacknowledged unreplicated data has to be waited for
- Expired timers have to be executed
- Interrupted socket calls have to be assumed

If the switchover happens while the active card is transmitting, the transmission must be re-done. That will be accomplished without additional code, as when no acknowledgment is received, retransmission will be done by the normal TCP retransmission mechanism.

As received data is not replicated until it is acknowledged, it may get lost during the switchover. No additional logic is required for it, as the remote peer will automatically retransmit unacknowledged segments.

Using the TCP retransmissions mechanism on the above cases to reduce the need for replication has one slight disadvantage – some connections may be terminated if a switchover occurs during a link failure. If a link has failed, the remote peer will try to retransmit periodically. When the failed link recovers, it is possible that a switchover coincides with a retransmission from the remote peer. The retransmission is lost. If the retransmission was the last to be sent, TCP connection is dropped by the remote peer. As on such situation connection is likely to break anyway (TCP continues retransmits for approximately two minutes), this is only a minor inelegance, not a problem.

As expiring timers do not necessarily cause replication, all timers that should have expired after the last replication must be executed. This can easily be accomplished by saving time-stamp of the last replication and comparing it to the current time. It would be possible to optimize this step away, as all externally visible events cause replication and the TCP timers do not have to be very exact – a timer can be delayed a few seconds without significant consequences.

There exist two separate scenarios for switchover that interrupt socket calls: socket call is in execution or socket call is blocking.

In the first scenario the active card is executing socket operation, but no replication has been performed to the passive (or replication was interrupted by the switchover) card when the switchover occurs. In this scenario the passive card just has to execute by itself the queued socket operation.

In the second scenario call blocks and the replication before blocking has been successfully performed. On such case no additional logic is necessary, as the waiting call will be wakened by the event it is waiting from the stack. The socket operation does not have to be re-issued.

11.2.7 Insertion of new control card

When booting with single control card, there is no need for replication. When a new passive control card is inserted in running node that already has a control card or the node is booted with two control cards, both the BGP-4 routing daemon and the protected TCP state machines have to be initialized in that card. As with method one, the bulk copy methods presented earlier are applicable. Replicating the TCP connection state machines requires no additional logic, as replicating does not differ from the regular replication.

It is important to guarantee that the BGP-4 routing daemons remain synchronized, when daemons at both cards are activated. As stacks are implicitly synchronized, they require no special attention. This can be accomplished with the following sequence:

1. Disable emitting any new events on the active card.
2. Perform replication.
3. Enable the BGP-4 stack on the passive card.
4. Enable the BGP-4 stack on the active card.
5. Enable emitting events on both cards.

Emitting events from the route database must be disabled to guarantee that BGP-4 input queues are identical in both the active and the passive card.

11.2.8 Signaling diagrams

Insertion of control card

Figure 11.7 shows the signaling diagram of the insertion of a new passive control card to a node where only one control card is originally present.

When the passive card boots, it first initializes both TCP and BGP-4 stacks. Unprotected TCP connections may proceed. BGP-4 daemon is suspended (no processing is started on the first hand). The passive card requests synchronisation from the active card.

The active card suspends events from the route database. The BGP-4 daemon is suspended, as is the protected portition of the TCP stack. Replication is performed for the TCP stack and the BGP-4 daemon. System software will synchronise the route database. The active card informs the passive that synchronisation is finished.

The passive card starts the BGP-4 stack and informs the active card that it is finished. This step is to gurantee that the BGP-4 daemon has on passive card all the events that the daemon on the active card has. Depending on the route database implementation it might be possible that there is no need to perform enabling events synchronously. The passive card then asynchronously enables the events from the route database.

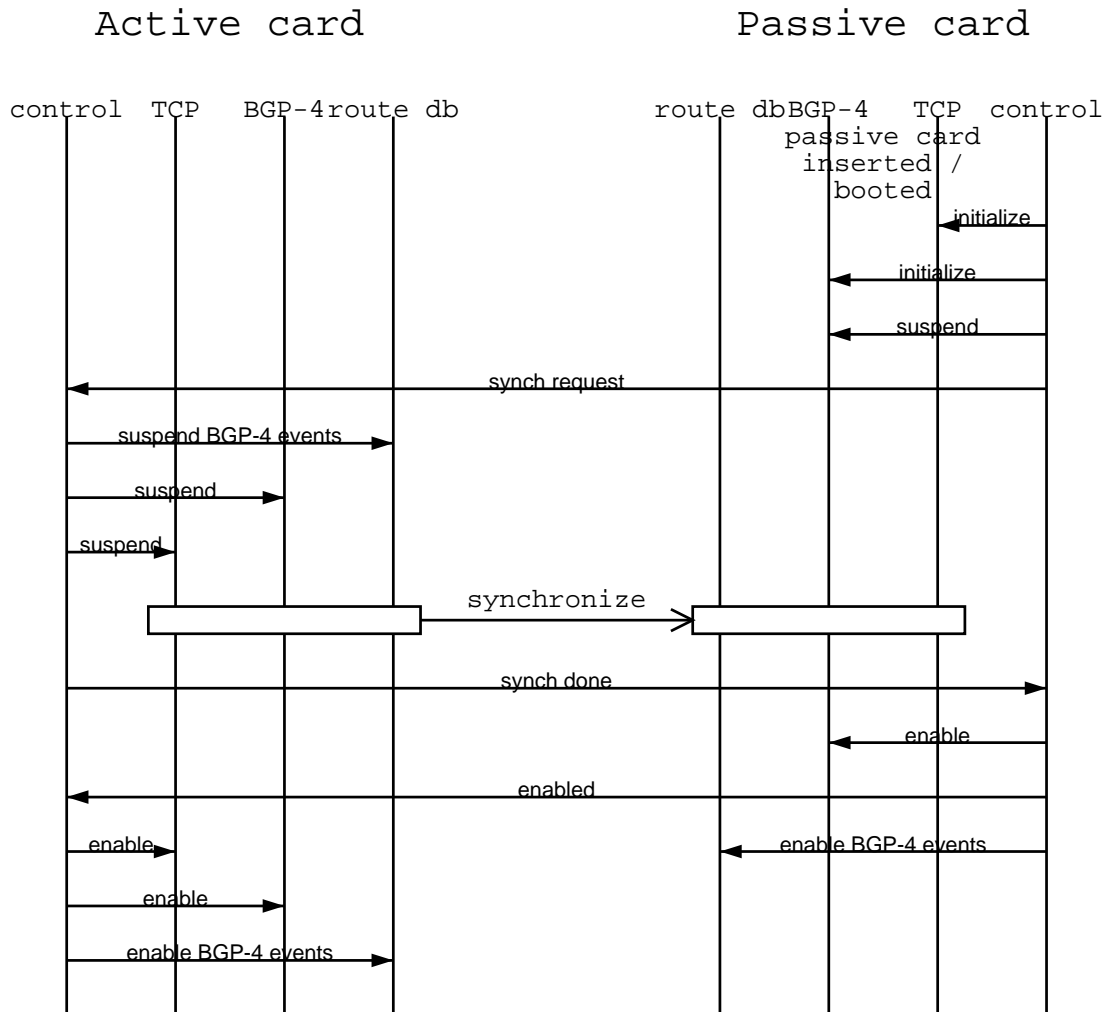


Figure 11.7: Insertion of a new card (method 2)

The active card enables the protected portion of the TCP stack, the BGP-4 daemon, and finally BGP-4 related events from the route database.

The sequence is similar with the similar situation on the method 1. The main difference is that this is somewhat simpler and the events in this signaling chart could quite freely be re-ordered.

Nonblocking send

Figure 11.8 shows a scenario where a BGP-4 timer initiates sending of a TCP segment (e.g. keep-alive). In this scenario TCP buffers have enough space and no blocking is necessary.

BGP-4 daemons receive timer tic on both cards. Processing timer is synchronized with message passing. The timer causes a packet to be sent. The BGP-4 daemon synchronizes send call with a message.

On the passive card the TCP stack recognizes that socket identifier refers to a socket that is as-

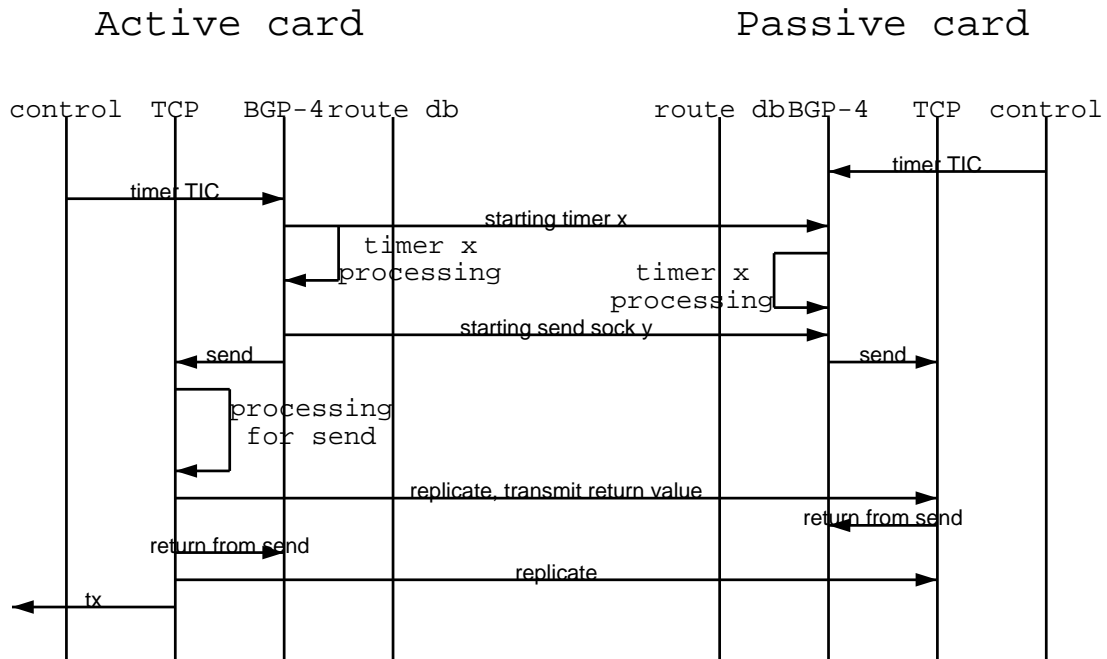


Figure 11.8: Nonblocking send (method 2)

sociated with protected TCP connection. The `send` call is queued. On the active card the TCP flow control is consulted and as enough buffer space is detected, data is put to the send queue. Typically TCP stack would transmit large segments out immediately, but on this scenario transmit has been delayed to make its independence of the socket interface more obvious.

When the `send` call is about to return, replication is performed. The return value of the socket operation is transmitted from the active card to the passive card.

Before the actual TCP segment transmit is performed, the new socket state is the replicated to the passive card. If transmit would have been executed within socket operation, replication would have had to flag the `send` call so that it would not have been mistakenly executed twice, if switchover would have occurred.

Blocking recv

The figure 11.9 shows a scenario where the BGP-4 daemon issues receive call that blocks.

The BGP-4 daemon uses message passing to synchronize the starting of the socket receive operation. The passive card only queues the `recv` call and does nothing else. The TCP stack on the active card consults receive buffers, and detects them empty. The TCP stack suspends the BGP-4 process until data is received from the network.

When a TCP segment belonging to the connection is received, the TCP stack decides to send immediate acknowledgment. Immediate acknowledgments are only sent when there is in transmit buffers data to piggy bag with acknowledgment. Usually acknowledgments are sent with a delay or with the

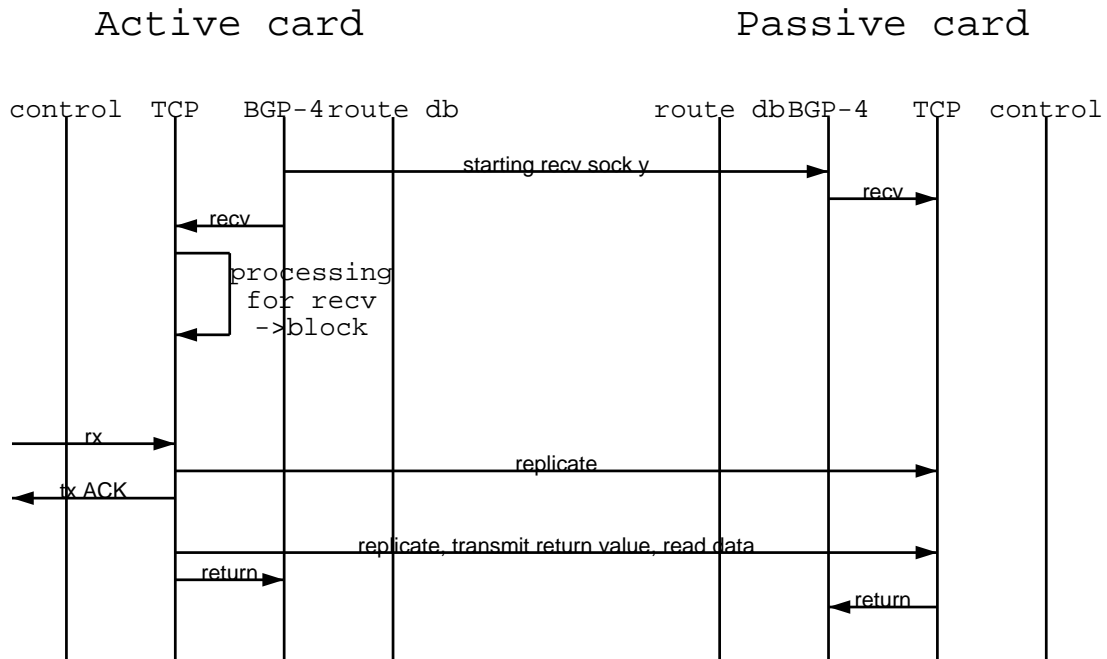


Figure 11.9: Blocking receive (method 2)

next data segment. Replication is performed to secure data that we are about to acknowledge. If no acknowledgment would be sent, this additional replication would not be executed. Acknowledgment is transmitted by the active card.

Proper return values and buffers are prepared for returning from the receive call. Replication is performed to update the state and to transmit return value to the passive card. Both TCP stacks return the received data.

Switchover

A scenario where switchover occurs in the middle of a socket call `send` is shown at figure 11.10.

In this scenario the BGP-4 daemon wishes for some reason to transmit data on a TCP connection. The start of the socket call `send` is synchronized with a message, and the BGP-4 daemons on both cards issue a `send` call to the TCP stack.

The TCP stack on the passive card queues the `send` request. The active card performs processing related to sending, but crashes before having an opportunity to transmit the actual packet or before returning from the socket interface call.

System software issues switchover notifications to the BGP-4 daemon and to the TCP stack on the passive card. The TCP stack restarts the queued `send` call, and performs processing necessary for `send`. The `send` call then returns and at some later moment the TCP stack decides to send actual TCP segment.

A more complex switchover scenario is presented on the figure 11.11. In it `send` call is interrupted

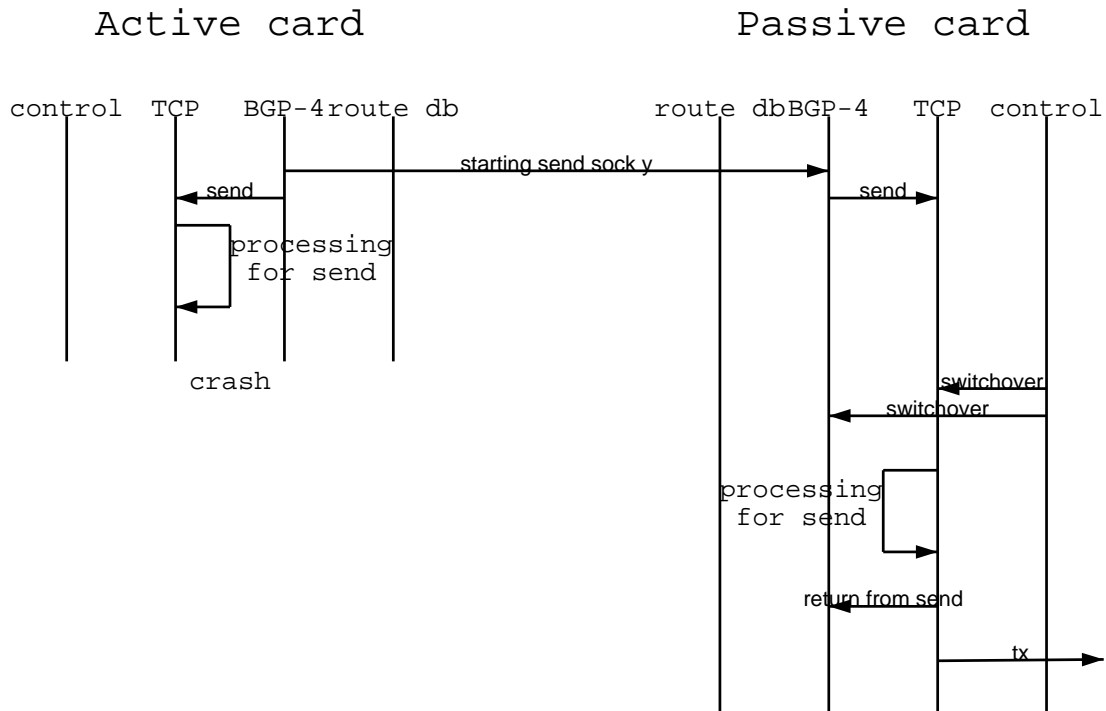


Figure 11.10: Switchover in middle of send (method 2)

in the middle of the TCP segment transmission.

The start of the scenario is identical to the previous scenario. For some reason, such as that send request is large, the TCP stack on the active card decides to transmit a TCP segment immediately. Before transmitting the TCP segment, replication is performed to update the passive card on the acceptable acknowledgment range. At the same time the queued send call is flagged being in process.

While transmitting the TCP segment, the active card crashes. In the example presented by the diagram the segment did not transmit fully, and is therefore ignored by the remote peer.

Switchover occurs, but as the queued send call is flagged as being in process, the only thing that is done, is to return from the call as if it had completed correctly. When no acknowledgment is received from the remote peer, normal TCP retransmit mechanism causes a retransmission to be performed. The normal TCP retransmission mechanism covers the data lost on interrupted transmit.

A scenario where the active card crashes in the middle of replication is shown on the figure 11.12.

In the scenario a new TCP segment from a remote peer is received. The active card performs normal TCP processing for the segment. The TCP stack waits for a while, and as no data is pending for sending, it decides to send a zero length TCP segment with acknowledgment. Before acknowledgment is transmitted, replication is necessary.

The active card crashes during the replication. Switchover is performed and the passive card discards the incomplete replica and reverts to previous replica of the TCP state. No pending timers or socket operations are detected, so nothing is done for a while.

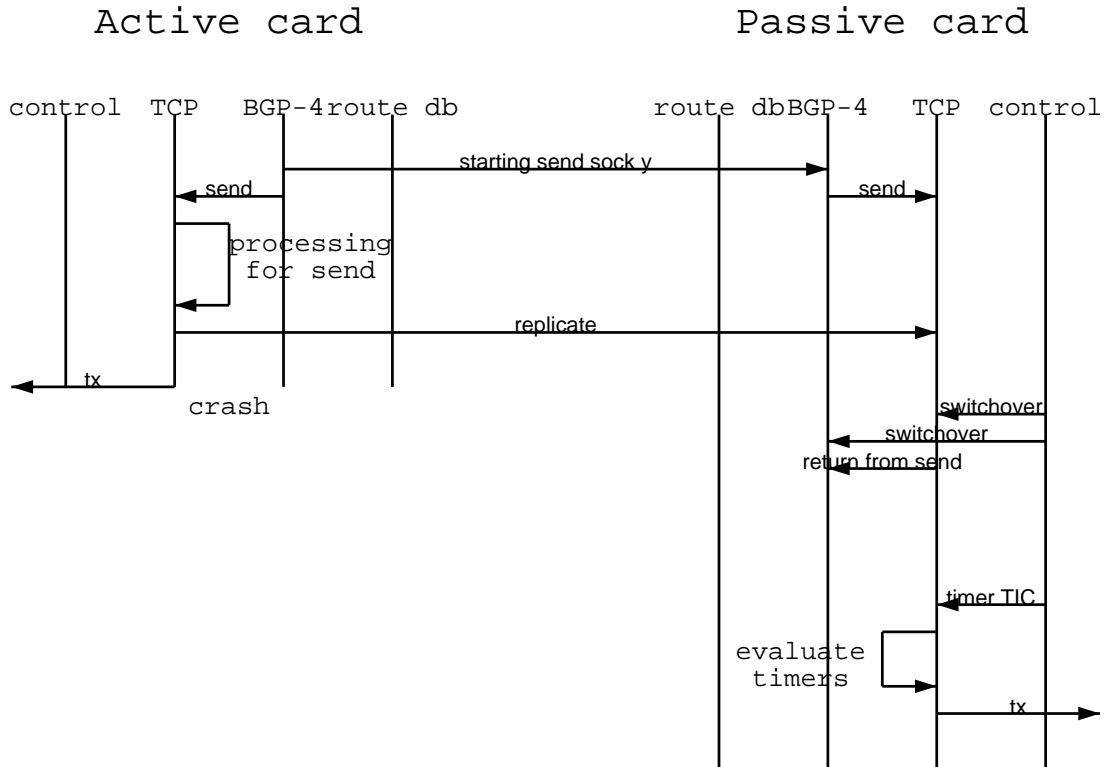


Figure 11.11: Switchover in middle of transmit (method 2)

The remote TCP peer does not receive acknowledgment and therefore retransmits the TCP segment again. This retransmission is caused by the regular TCP retransmission process. TCP processing is carried in normal fashion on the protecting card. After a while an acknowledgment is sent to the remote peer. The TCP stack is now identical to what it would have been, if no crash would have occurred.

11.2.9 Advantages and disadvantages

One of the key advantages of this method is, that modifications to the actual TCP implementation are relatively few, although probably larger than with method 1. The associated disadvantage is that large modifications to socket implementation are necessary.

Another advantage for this method is that the method is easy to understand and simple. As the initial synchronization is identical (except for size) to the regular synchronization, no additional logic is necessary for it.

This method requires less synchronization and therefore it could be marginally faster than more synchronized methods.

The method does not utilize hardware based packet duplication facility on the node, and therefore replication seems somewhat redundant. Although the link between control cards has plenty of extra bandwidth, the memory bandwidth of the nodes is limited. The memory bus bandwidth may become

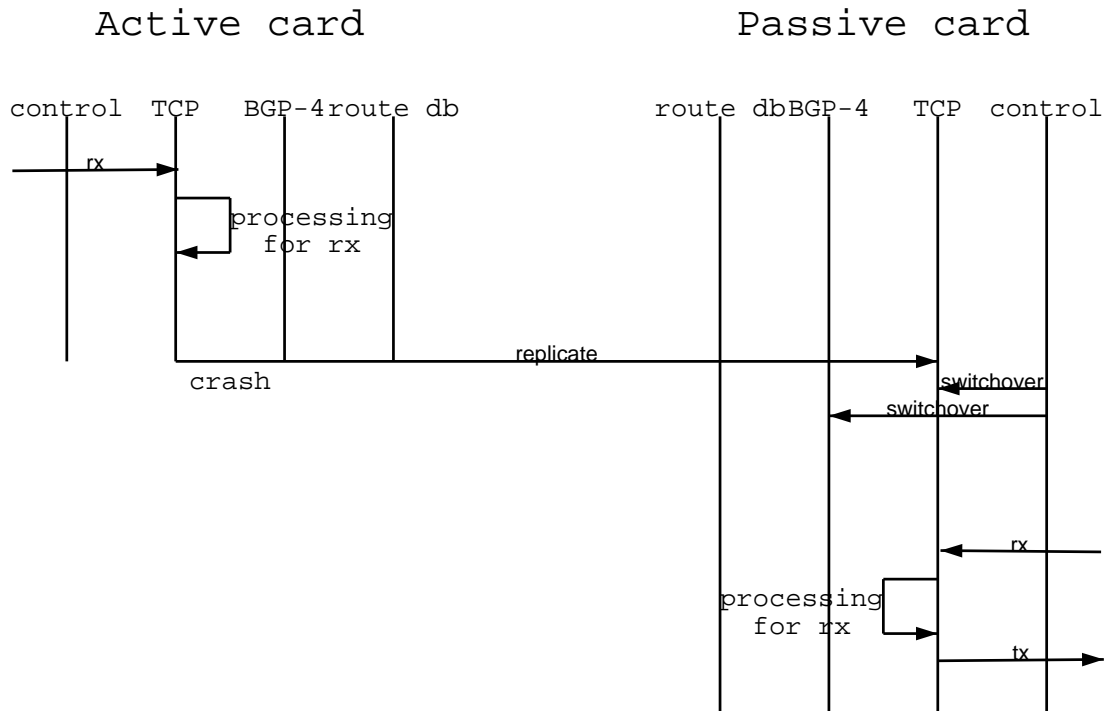


Figure 11.12: Switchover in middle of replication (method 2)

performance bottleneck.

This method is applicable for other TCP applications, as long as all protected applications perform socket operations in the same order. As with method 1, this requirement limits applications of the protection in the field of software upgrade.

As the socket structures are identical on both cards, task identifiers must be synchronized for the task using protected TCP connections.

11.3 Method 3: replicating TCP and BGP-4 state

In this method neither the protected partition of the TCP stack nor the BGP-4 daemon are executed on the passive card. The state information of both stacks is replicated from the active card.

The picture 11.13 shows a general overview of this method.

As can be seen from the picture, this method does not differentiate between initial and following replications – all protection is performed with replication. This method is very lightweight, if MMU assisted copying is used as the bulk transfer method.

Only external interfaces of the protected system are the interface towards the system route database and towards the IP stack. Therefore rules when to replicate are relatively simple. The replication must be performed:

- Before a TCP segment is sent.

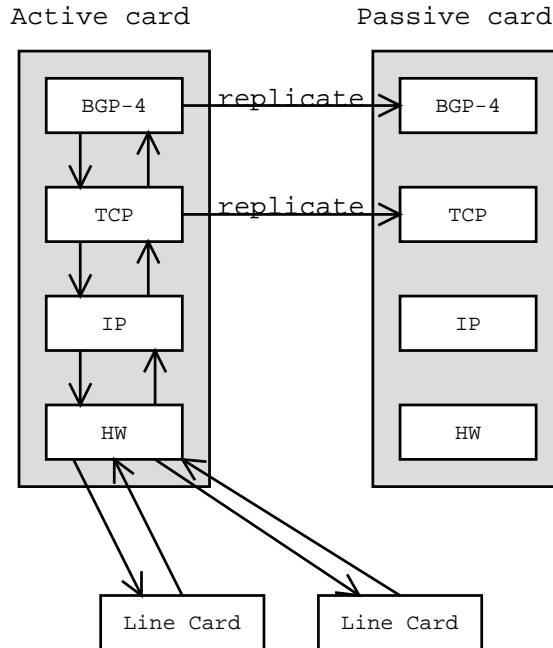


Figure 11.13: Overview of method 3

- Before an event received from the system route database is acknowledged.
- Before an event is sent to the route database.

These rules are based on the principle that non-protected entities must not notice possible switchover. This method considers the BGP-4 daemon and the protected TCP stack as a single whole and replication is always performed for both. If replication were only performed for one of those two components, there would be need for complex rules to ensure consistency at socket interface.

11.3.1 Transmitting TCP segments

When a TCP segment is sent, the sender also commits to accept acknowledgments for the sent packet. Therefore the replication has to be performed after the TCP stack has performed processing for the send, but before the actual buffer is passed to the IP stack send queue. This arrangement guarantees that if switchover happens after transmitting the segment, the passive card is willing to accept acknowledgment for it. As with method 2, this model utilizes the normal TCP retransmissions to recover packets lost in switchover that occur between the replication and actual transmit.

Replication must not be skipped before sending zero length TCP segments. This behavior guarantees that replication is always performed before sending a TCP segment with ACK bit set (same applies to acknowledgment by the selective acknowledgment option). Replication before sending an acknowledgments is vital, as sending the acknowledgments will cause the sender to remove acknowledged segments from the transmit buffers and therefore removes the possibility of re-obtaining data later.

11.3.2 Receiving TCP segments

No special behavior is necessary for receiving TCP segments. As presented in the previous chapter, replication is performed before sending an acknowledgment for the received segment. Normal TCP retransmissions are used to recover packets lost in the interval between the last synchronization and the activation of the passive card.

11.3.3 Socket interface

No modifications are necessary to the socket interface, as it is contained within the protected system. This assumption sets the limitation that all protected TCP applications must be protected with this same method.

11.3.4 Routing database interface

This protection method sets a special requirement for the system route database protection: route database on the passive card considers an event to be sent only after the route database on the active card has received acknowledgment for the sent event. Basically this means that all events for BGP-4 daemons sent by the system route database wait for an acknowledgment message. Purpose of this extension is to ensure that replicated copies of BGP-4 state always match with the state of the system route database and no event gets lost during the switchover.

Replication is always performed when an event is received from the system route database. The event is acknowledged when replication has been performed, and the event is processed normally.

Replication is also performed before sending an event to the system route database. Purpose of this replication is to guarantee that no event sent by the BGP-4 daemon is lost during the switchover.

Both BGP-4 daemon and system route database have to ensure that no duplicate event is processed. The replication strategy itself guarantees that no message is lost, but it does not guarantee against duplicates caused by the fact that the system route database is not necessarily replicated at the same time as is the BGP-4 daemon. Avoiding double events is easily accomplished by numbering events.

An easier alternative would be to specify that the route database and other routing protocols must be synchronized at the same time as the BGP-4 daemon. This approach would be the easiest to implement, but is not presented in this thesis, as no detailed information of the system route database is assumed.

11.3.5 Replication

Replication is performed with one of the bulk transfer methods presented earlier. As replications are frequent, DMA based differential memory transfer is the best suited bulk transfer method for this protection method. It should be noted that this method is feasible only because of the convenient extra bandwidth available between control cards.

There is no inherent difference between initial and subsequent replications, although it is possible to transfer only changed structures on the subsequent replications. Obviously processing incoming IP packets belonging to protected connection, evaluation of timers and receiving events from the routing database have to be suspended during the actual copying.

Some form of mutual exclusion must be utilized to guarantee that replication is performed at such time that the state of both the TCP stack and the BGP-4 stack can be recovered. If no modifications are done to the stacks, the rule is that when replication is performed, the stack that did not cause replication must be in the main loop. For example if the TCP stack requests replication, replication may begin only after the BGP-4 daemon has entered to the main loop. In most implementations the rule can be relaxed, as the only thing that is really required is that the other stack is in such state that can be expressed with the replicated data. With modifications to the stack, it might be possible to remove this requirement altogether. Obviously the stack that initiates replication must be in such state that the replica can recover itself. These requirements are obvious when one considers the fact that although the protocol state machines are replicated, actual program counters and call stacks are not.

The rules above have an interesting consequence: the BGP-4 daemon must use non-blocking socket calls. This is not a problem, as most of BGP-4 implementations use asynchronous IO. Asynchronous IO is often used, as blocking IO is not feasible when peer count for single router can be even thousand. Although asynchronous IO does not block, `select` call does. Therefore `select` calls must be checkpointed so that replication can safely be performed even if the BGP-4 daemon is blocking at `select` call.

BGP-4 routing traffic is bursty by its nature – although HELLO messages are periodic, other messages are only exchanged during initial database synchronization and when the network topology changes. When topology of the network changes, or initial database synchronization is performed, huge amounts of one way traffic is transferred. An obvious optimization is to advertise a large TCP window, and delay sending acknowledgment. Such additional delay would cause all traffic in single burst to be processed with two replications. First replication would of course be before sending an acknowledgment and second would be before informing system route database of changes.

11.3.6 Switchover

As replication is only performed when an external input is received or output is emitted, the replica is likely to contain many pending timers. At the moment of switchover all of these timers have to be evaluated at hastened phase. Hastened evaluation serves two purposes: it ensures that all events are performed when they should be and on the other hand they cause retransmission of packets that may or may not have been correctly transmitted before the switchover.

The system route database needs to re-emit all unacknowledged routing events. Both the system route database and the BGP-4 daemon have to discard all duplicate events they receive.

No other action is necessary, and possible lost packets are handled by the normal TCP retransmission mechanism.

11.3.7 Signaling diagrams

Insertion of control card

Insertion of an additional control card to a node is shown in figure 11.14. Before insertion the node already had one control card, and the newly inserted card is in the role of the passive card.

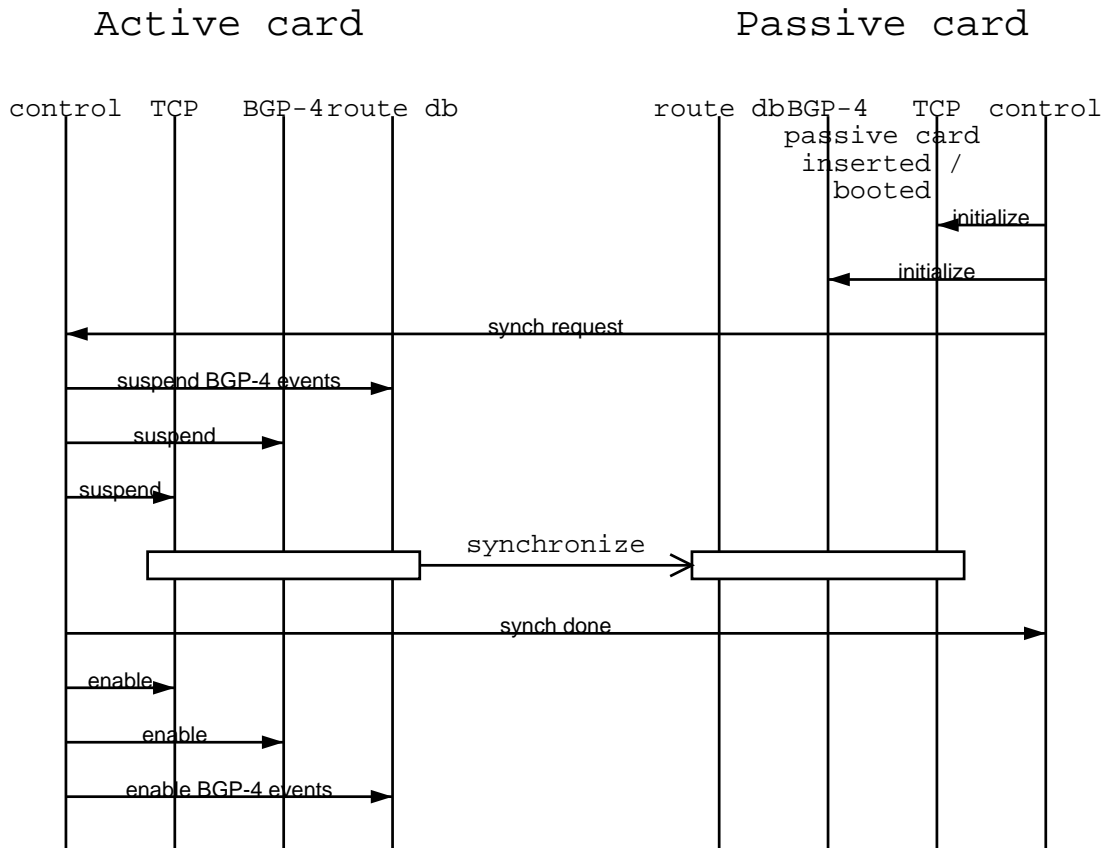


Figure 11.14: Insertion of a new card (method 3)

When the passive card boots, it initializes both the BGP-4 stack and the TCP stack. In the normal case the purpose of these initializations is to allocate and initialize unprotected data structures and enable unprotected TCP connections. If a switchover would occur during the initial synchronization, these bare bones memory images would be taken in use instead of the half-complete images transferred from the failed active card.

When initialization has been performed, the passive card requests initial replication. The active card suspends processing of events from the system route database, evaluation of timers in both TCP and BGP-4 stacks, and finally processing of incoming TCP segments. The replication is performed with any of the bulk transfer methods.

When replication has been performed, the active card enables all previously suspended processing. Unlike the previous methods, synchronization with the passive card is necessary, because nothing is

enabled on the passive card. The node is now capable of performing switchover without disrupting the level of service.

Send

The figure 11.15 shows a scenario where a route learned by and IGP is advertised further with protected BGP-4 connection.

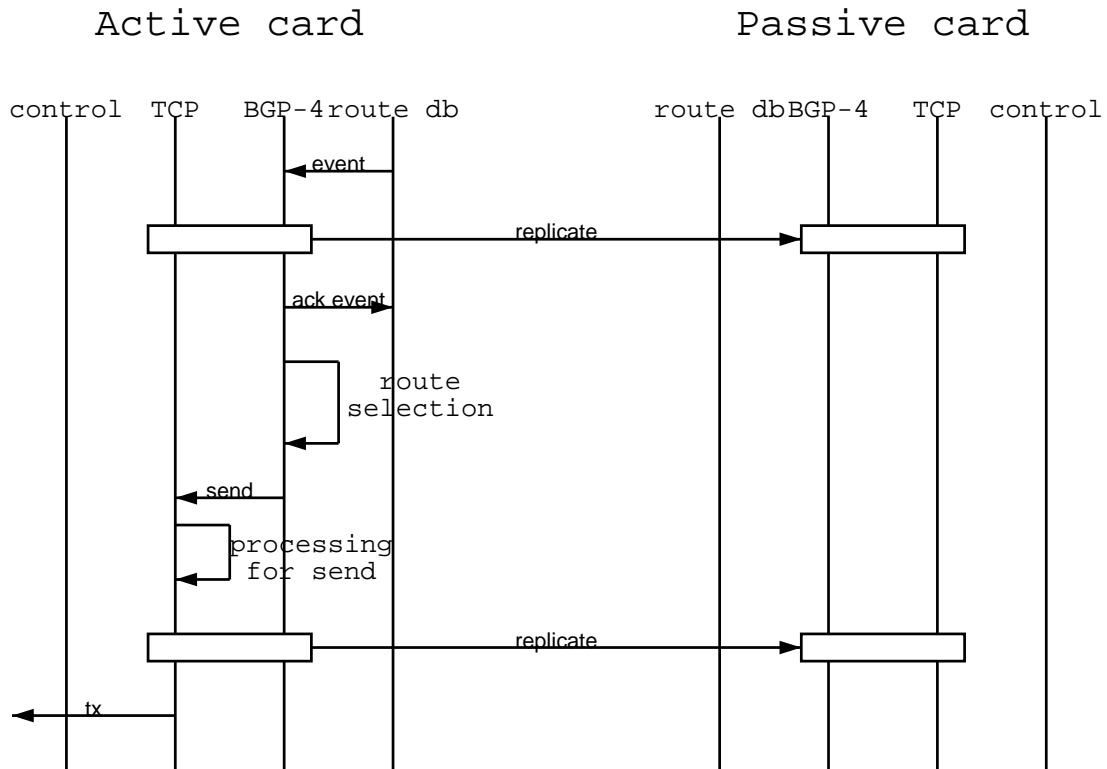


Figure 11.15: Sending TCP segment (method 3)

In this scenario some IGP daemon learns a good route to some destination. When the system route database receives information about this new route, it sends an event to BGP-4 daemon on the active card. On the active card the route database sends an event to the BGP-4 daemon. On the passive card no event is sent.

The BGP-4 daemon adds the route to its RIBs and performs replication to the passive card. The event is then acknowledged. The BGP-4 daemon performs route selection to calculate whether or not this new route should be advertised. In this scenario the new route is advertised to single peer, and a BGP-4 packet is constructed and send socket call is executed.

The TCP stack performs necessary processing for the packet, enqueues it, and returns from the socket call. Later the TCP stack decides to send a TCP segment containing some of the buffered data. The segment is created and proper headers are attached to it. Replication is performed before the actual send is commenced. Then the new TCP segment is transmitted to the network.

Receive

A typical TCP segment receive scenario is shown on figure 11.16. This scenario presents a typical situation where the BGP-4 daemon waits for incoming BGP-4 packets.

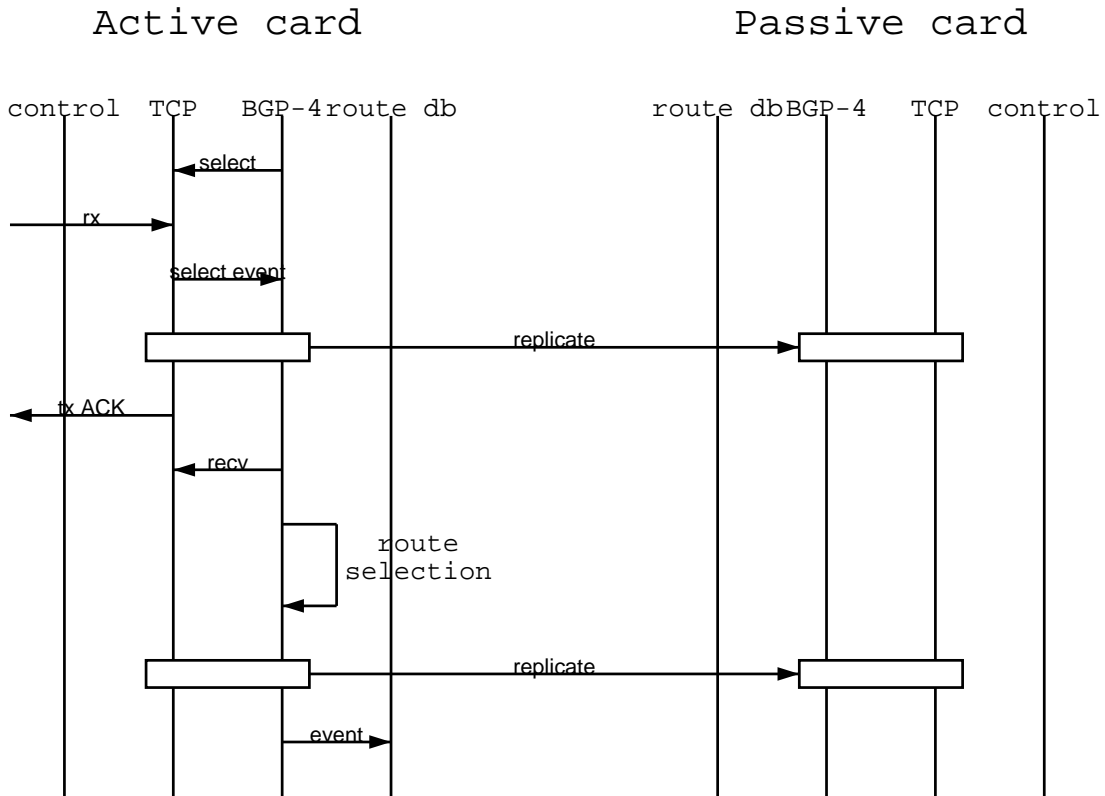


Figure 11.16: Receiving TCP segment (method 3)

The BGP-4 daemon wishes to be informed about received packets and for that end issues `select` call. A TCP segment is received from the network. The segment contains new data that is added to receive buffers. Acknowledgment for the segment is hold back in the hope of piggybacking it with a data segment.

The BGP-4 task is woken and the `select` call returns indicating that a new data is readable.

As there is no data to piggyback acknowledgment with, the TCP stack decides to send an acknowledgment for the received data. Before the acknowledgment is sent, replication is performed. Then the acknowledgment is sent.

The BGP-4 daemon issues nonblocking `recv` call to retrieve received data from the TCP stack. The received BGP-4 packet contained information that forces the route selection procedure to be performed. As a result of route selection, the system route database needs to be updated.

Before an event is sent to the system route database, replication is performed to both guarantee that no event is lost and to reduce possibility that the expensive route selection would have to be executed at switchover situation. In theory this replication is unnecessary, but in practice it makes switchover

much easier. The event is then sent to the system route database.

Switchover

This protection method has two special cases of switchover that merit further examination. These special cases occur when a switchover occurs during processing of a route database event or during transmitting IP packets.

The figure 11.17 shows a scenario where switchover occurs during a route database event.

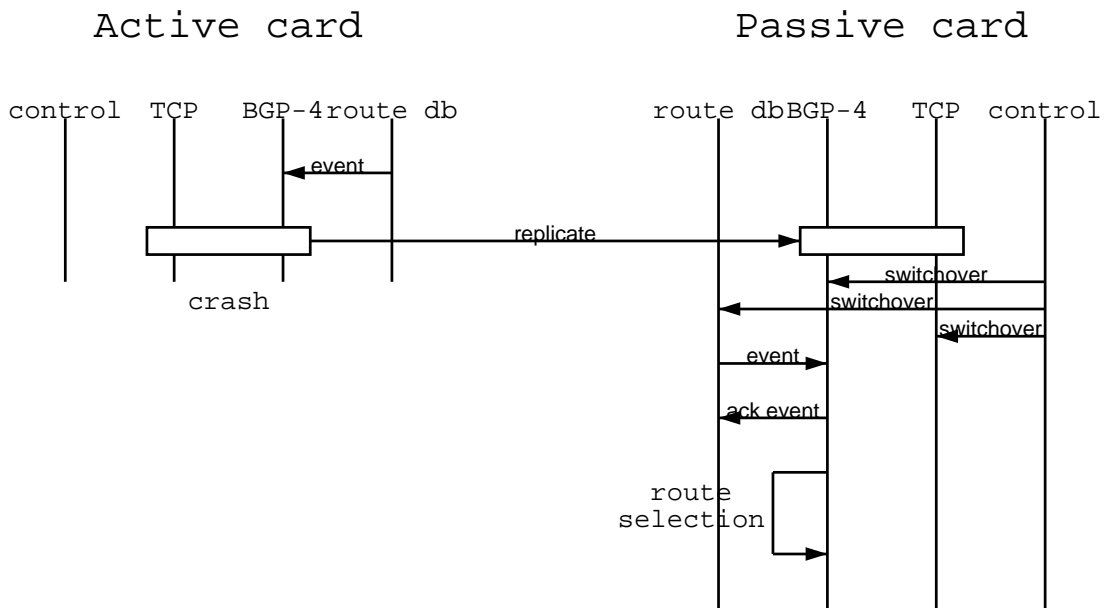


Figure 11.17: Switchover during processing of route database event (method 3)

The system route database on the active card sends a routing event to the BGP-4 daemon. The daemon adds event information to the local database and performs replication to the passive card. Due to a hardware failure the active card crashes during the replication. The incomplete replica is discarded and the previous one is used. The previous replica is the replica that was created by the previous replication caused by any event or if the crash occurred at the first event, the replica that was created during the initial replication.

Switchover is performed and the passive card becomes the active one. A switchover notification is sent to the BGP-4 daemon, to the system route database and to the TCP stack. The order of these notifications does not matter, as replication rules above do not allow socket calls to be interrupted.

The system route database notices an unacknowledged event and re-emits it immediately. The BGP-4 daemon has old state information and is therefore not aware of the previous instance of the same routing event. Therefore the routing event is processed normally. The routing event is then acknowledged. No replication is performed, as there is no protecting card when the other card is down.

The BGP-4 daemon then performs route selection. The change indicated by the routing event has not caused any changes to the advertised routes and therefore no packets are sent to the peers.

The other special case of switchover is presented on the figure 11.18.

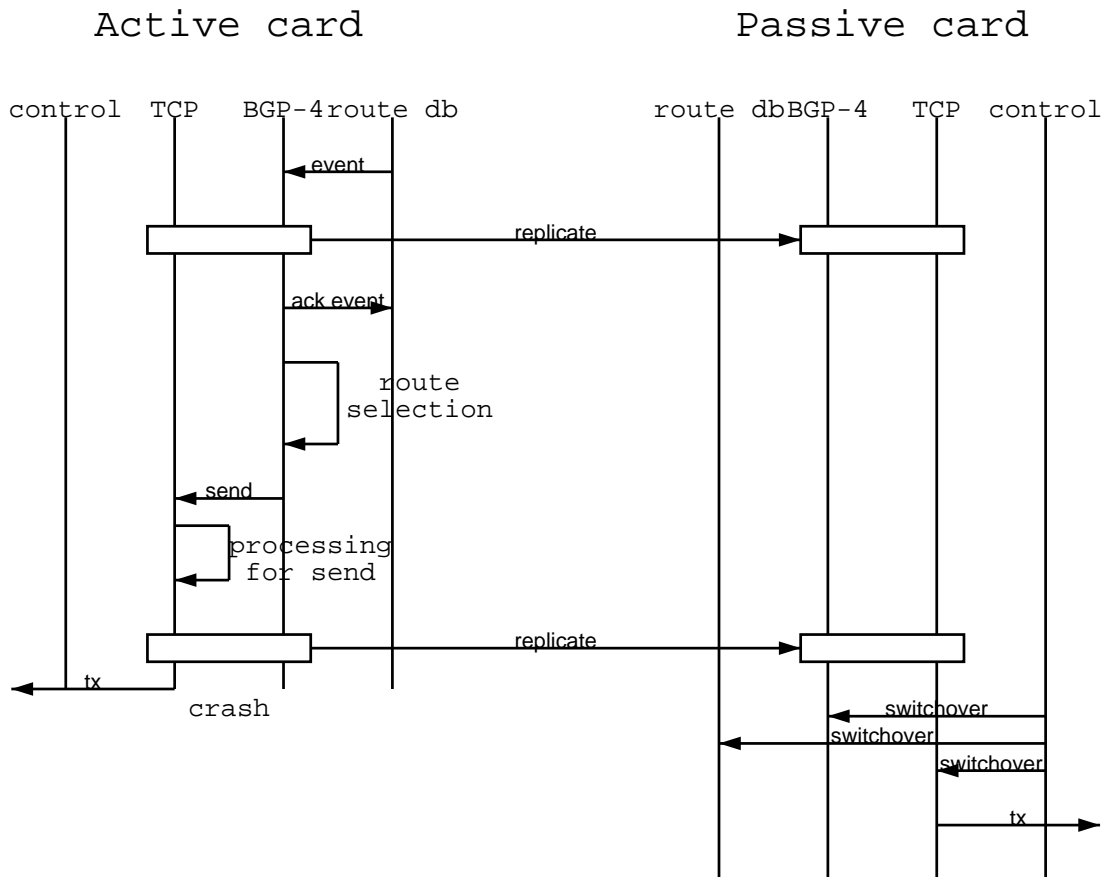


Figure 11.18: Switchover during transmission (method 3)

The scenario starts similarly with the send scenario presented above – an event from the system route database causes a change to routes to be advertised. The change in advertised routes affects one BGP-4 peer, and the BGP-4 daemon issues send call to the TCP stack. The TCP stack performs necessary processing for the send. Before the transmission of a TCP segment, replication is performed to protect changed state on the socket interface.

During the actual transmission the active card crashes. The TCP segment was only partially transmitted, and therefore has incorrect checksum. The partial TCP segment will be discarded by the remote peer.

Switchover occurs, and all components on the passive card are notified of it. The order of notifications makes no difference. In actual implementation it is likely that checkpoint mechanisms would require the BGP-4 stack to be enabled first. Depending on the implementation the interrupted transmission of the TCP segment is either continued immediately or when a retransmission timer expires. The diagram assumes that transmission is performed immediately.

If switchover would have occurred during the replication before the transmission, the replica made during the route event processing would have been used. In that case socket call to the TCP stack would have been re-issued. If switchover would have occurred right after the transmission, unnecessary retransmission would have been performed by the passive card. As the remote peer ignores unnecessary retransmit, no harm would have occurred.

When the TCP segment has been transmitted, the passive card is in the state the active should have been after successfully performing the transmission.

11.3.8 Advantages and disadvantages

The most important advantage of this protection method is that modifications to the BGP-4 daemon and to the TCP stack are relatively few. If only non-blocking IO is necessary, replication calls to a few selected positions are all that is necessary. On the other hand if blocking socket calls are necessary fine grained checkpointing of the BGP-4 daemon is necessary. Of course the memory allocator has to be modified, but in practice modifications are minimal.

An advantage that this method shares with the method 2 is that it is easy to understand. There is no complex initial replication, as all replications are the same.

As this method does not require synchronization, only replication, it is feasible to expect higher performance than with methods one and two. This expectation is based on the fact that replicated datasets are relatively small while the bandwidth of the link between the control cards is huge. A limiting factor for the performance is how fine grained synchronization is necessary between the BGP-4 daemon and the TCP stack. If more fine grained synchronization is used for selecting replication time, performance is better, but on the other hand modifications to the BGP-4 code are larger.

Hardware based packet replication is not utilized, and therefore this method seems to be a poor fit for our environment. On the other hand the link between the control cards is utilized as much as possible. There is some risk that memory bandwidth utilization of the active card might reduce performance drastically when certain traffic level is reached.

The use of this method tends to create situation where work load is unevenly distributed between the active and the passive card. Uneven load distribution is good, if the active card has enough CPU power to perform all of its task – the passive card is mostly idle and that saves power and therefore eases problems related with the cooling. On the other hand if the CPU of the active card is not powerful enough, more even load distribution is necessary.

Although this method can be used for all types of TCP applications, the lack of support for blocking sockets hinders general usability of this method.

11.4 Method 4: graceful restart mechanism for BGP

The graceful restart mechanism for the BGP-4 introduced in chapter 8.1.1 may become internet standard, and therefore should be seriously considered.

Although the graceful restart mechanism is intended for single control card environment, there is no need to abstain from using it in two control card environment. In two control card environment no synchronization or replication is necessary between the control cards. Instead when switchover occurs, the BGP-4 daemon on the protecting card re-establishes all connections and re-obtains all BGP-4 routing information. As the state of forwarding plane is retained during the switchover, the protecting card can pretend to be the former active card restarting.

The graceful restart extension is unable to provide protection in situations where switchover occurs in middle of forwarding plane update. On such situation the forwarding plane would go to inconsistent state. As the forwarding plane consists of line cards with separate processors, it is easy to arrange the update to happen atomically – modifications are transmitted to line card and activation command is sent as multicast. This arrangement guarantees that the graceful restart extension works in every case.

As the graceful restart extension is extensively introduced in chapter 8.1.1, it is not described in detail here.

11.4.1 Switchover

The switchover procedure is always the same, regardless of what the active card was doing at the moment of the switchover. Figure 11.19 shows the switchover in a situation where there is originally only one BGP-4 peer. Switchover with multiple BGP-4 peers is identical to this simple case, but in it multiple connections are re-established simultaneously.

When the switchover occurs, the BGP-4 daemon and the system route databases are notified of it. The TCP stack needs not to be notified, as it contains no protection logic. The system route database marks all BGP-4 routes as stale. The BGP-4 daemon performs start-up procedures and requests TCP stack to establish a connection to the peer. In the figure the connection attempt succeeds. It is also possible, that the remote peer manages first to connect to us. In both of the above cases the result is a full duplex TCP connection.

The BGP-4 peers exchange OPEN packets. The passive card uses restart flag in the graceful restart capability option to indicate that the router restarted – switchover is restart from the perspective of the graceful restart extension. The forwarding bits are set to indicate that there was no disruption to the forwarding.

The remote peer sends its database to the passive card. When the end-of-RIB marker is received, the route selection is performed. Resulting routes are sent to the system route database. The system route database removes stale routes and instantiates these new routes to the forwarding plane. In stable networks no route changes happen during the switchover and therefore this procedure often produces no changes to the forwarding plane.

The passive card then advertises its routes to the remote BGP-4 speaker. When the passive card sends end-of-RIB, switchover has been finished.

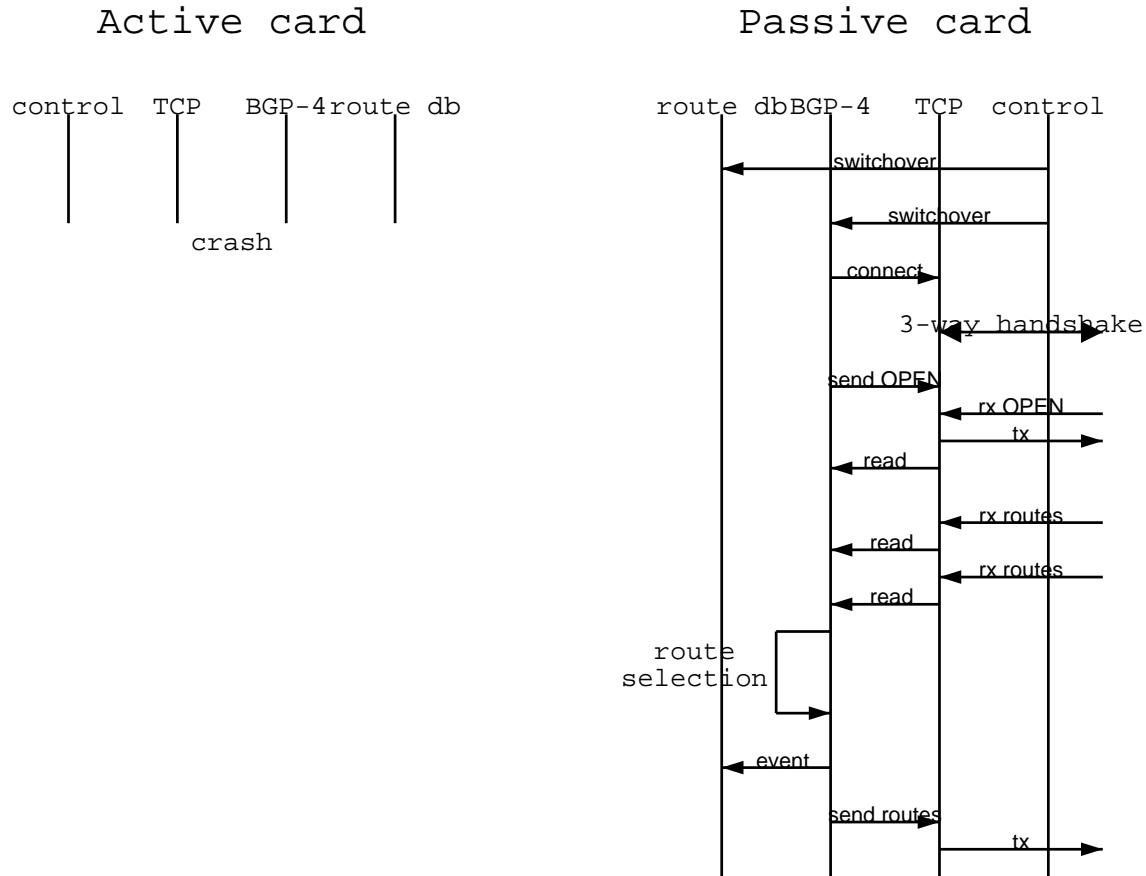


Figure 11.19: Switchover (method 4)

11.4.2 Advantages and disadvantages

This method is the easiest of all protection methods to implement and understand. No changes are necessary to the TCP stack and modifications to the BGP-4 daemon are limited to a clearly defined protocol extension.

The method causes no additional processing or bandwidth requirements and it does not affect the normal routing performance. As no data is transferred between BGP and TCP stack of the active and of the passive card, there is no need to worry about software version incompatibilities. Software versions just need to be compatible with the forwarding plane interface.

During the database reobtaining procedures following the switchover the router is unable to change forwarding that is based on the routes learned by the BGP or advertising changes in routes to other BGP-4 peers. This limitation may cause transient routing loops and errors during the recovery.

This method does not work with BGP-4 nodes that do not support the graceful restart capability. It remains to be seen whether or not this is a problem – the big players such as Cisco and Juniper are originators of the graceful restart draft and therefore it is likely that they will implement it. On the

other hand it is possible that the draft will be abandoned and most routers won't support the graceful restart mechanism. In any case, implementation of this method does not cause incompatibilities, as use of the protection is negotiated separately for each connection.

The use of this protection method can turn to be as an advantage or a disadvantage – if this method becomes a widely used, use of it enables protected service in multi vendor networks. On the other hand if this method is abandoned, use of it is possible only in single vendor networks.

The method is not applicable for protecting other TCP applications, as protection is achieved with modifying the BGP-4 protocol itself. The trend seems to be to specify similar enhancements to protocols. The label distribution protocol (LDP) already has such extensions.

11.5 Method 5: terminating TCP sessions at line cards

The obvious method for solving problem with TCP sessions is to terminate them at line cards. The TCP state machines are executed on the line cards and data is transferred with ITC multi casts to the both control cards. The situation reminds SOCKS servers used with some firewalls [16]. The arrangement is shown in figure 11.20.

As can be observed from the overview picture, the line card terminating the TCP session becomes a single point of failure for the connection that it is protecting. Line card protection becomes impossible, as use of it would require similar TCP protection methods between the line cards.

Load balancing is impossible, unless TCP packets belonging to a given TCP session are forwarded to a dedicated line card that handles the TCP session termination for that connection. In such situation failure of one line card would cause disruption to the all sessions handled by the failing card.

Although this method has severe disadvantages, it also has an appealing advantage: it is simple to implement. Most likely a modified SOCKS server and library could be used for implementation. Modifications to the SOCKS protocol would be necessary for implementing ITC multi cast support and support for multiple SOCKS servers (multiple line cards). Line cards correspond one or more network interfaces from the point of view of the SOCKS server. The correct SOCKS server for particular socket operation would have to be selected based on destination IP address. Accepting connections would have to be performed on all SOCKS servers. [16]

This method is unsuitable for hardware architecture that has several line cards, as on such architecture it only moves the protection problem to another location or creates a single point of failure. As backplane is passive it can not be used instead of line cards.

This method was presented only for the sake of completeness. Although the method is excellent for certain architectures, it is useless for highly redundant architectures, such as the one used in the carrier class router platform. This method must not be used.

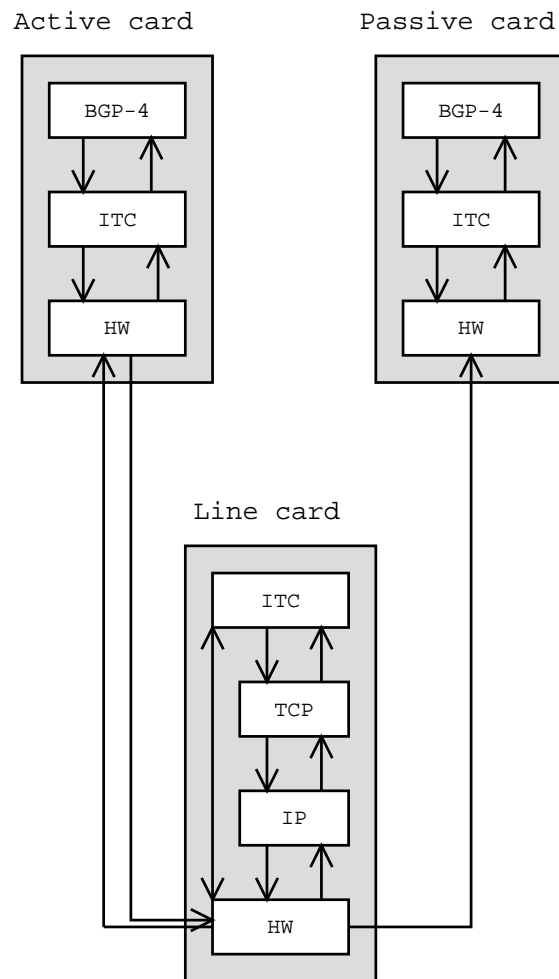


Figure 11.20: Overview of method 5

Chapter 12

Comparison of the protection methods

As described above, protecting all BGP-4 connections is vital. If BGP-4 connections are lost, no packets are sent to the router and all other protection measures are in vain. Below, the most suitable protection method is selected from the methods presented in previous chapter.

12.1 Selection of protection method

The methods presented in the previous chapters present four different approaches to protection: protocol modification, replication, synchronization and combination of replication and synchronization.

The method five is discarded from the selection process, as on our hardware architecture the method accomplishes nothing.

The chapters below evaluate the remaining four methods against the requirements we set forth in chapter 9 and consider other important aspects that affect the selection. Table 12.1 shows summary of evaluation of the protection methods.

Criteria	Method 1	Method 2	Method 3	Method 4
Minimal interruption to the traffic	none	none	none	some
Software version cross compatibility	partial	partial	partial	full
Compatibility with third party routers	full	full	full	circumstantial
Suitability for other TCP applications	partial	partial	partial	none
Changes to existing code	huge	large	large	minimal
Overhead	some	some	some	none

Table 12.1: Summary of protection methods.

The detailed descriptions of the methods 1, 2, 3, and 4 is available at pages 51, 62, 73, and 82.

12.1.1 Minimal interruption to the traffic

As forwarding is performed on separate hardware, the switchover in itself does not cause interruption to the forwarding of packets. Possible interruption occurs either because the forwarding plane contains

outdated routes or other routers choose not to forward data to the protected router.

Methods one, two and three are capable of performing the switchover in less time that it takes to process a single BGP-4 packet. Therefore, these methods are totally transparent to both BGP-4 peers and to the forwarding plane.

Method four may cause transient forwarding loops, and therefore the passing traffic may be temporarily interrupted. Although the loops occur only if network topology changes before the switchover is complete, some interruption to the forwarded traffic is possible. The method four does not fulfill this requirement fully, but provides adequate service for most applications.

12.1.2 Software version cross compatibility

As method four transfers no data between the TCP stacks or the BGP-4 stacks, there is no question of the suitability of the method for software update purposes.

The three other methods transfer varying amount of data between control cards. As in initial replication whole state information is transferred, incompatible software versions can break these three protection mechanisms. Although some incompatibilities can be restricted with conversion routines, conversion from the newer version to an older one is not necessarily always possible. Also, if new BGP-4 capabilities are added, it is entirely possible that switchover can not be performed from a newer version to older versions. If a new BGP-4 capability is negotiated with a remote peer, a card running old software version that does not support the new capability can in no way successfully talk to remote peer on the connection that has the new capability enabled. These problems can of course be limited by enabling only those BGP-4 features that both control cards can support.

Partial compatibility for software updates exists in the first three methods. This compatibility can be enhanced, with the cost of extra code and CPU consumption.

12.1.3 Compatibility with third party routers

The first three methods are transparent to other routers and therefore they do not require support from other vendors. Although these three methods protect the router, they offer no services for other routers to implement protection services.

Method four requires support from third party routers. The flip side of this requirement is that an implementation of method four offers services that third party routers can use for protecting themselves. In other words the router requires help for successful protection, and in return it can also help other routers to protect themselves.

Usually the whole network needs to be protection aware, and therefore it is vital that protection support is offered to other routers whenever it is needed. At the moment it is impossible to say whether or not method four takes wind under its wings, and therefore it can only be concluded that all of these four approaches have equal merits in this area. All of these methods possess a risk of possible incompatibilities pending how the standardization work proceeds.

BGP-4 is often used to exchange routing information across administrative domains. The operator can not dictate what equipment other operators and clients use. Therefore it is likely that even if method 4 would become an internet standard, routers that do not support it would be around a long time.

12.1.4 Suitability for other TCP based applications

The first three methods are suitable for other TCP based applications. Methods one and two require enhancements to protected applications to ensure identical ordering of events in both cards. The first three methods require that the protected applications are deterministic. Modifications for the replication are necessary for the applications that are protected with any of these three methods.

Method four is only suitable for protecting BGP-4 protocol. Similar enhancements can of course be specified for other protocols. The concept is re-usable but the implementation is not.

12.1.5 Minimal changes to existing code

Of these four methods, method four requires least changes to existing code base. The changes required by method four are small and limited to a small section of the whole code.

Method one requires complicated changes to both the BGP-4 daemon and to the TCP stack. Methods two and three require lesser amount of changes, but still much more than method four.

Method four is clear winner on this category.

12.1.6 Minimal overhead

Method four imposes almost no overhead at all compared to unprotected BGP-4.

Method one uses hardware based packet duplication service and causes only little overhead in a form of synchronization code. Although the overhead is minimal, the constant synchronization imposes delays to application and therefore reduces maximum possible performance of the BGP-4 routing daemon. The synchronization penalty does not affect to other processing on the cards.

Methods two and three impose some overhead in form of replication and synchronisation. With proper implementation and use of a good bulk copy method the overhead is relatively small and shows mostly on the memory busses of control cards. These methods have better performance than method one.

Method four is best on this category.

12.2 Selected method

Methods two and three appear to be usable, but are not as good as other methods. As hardware based packet replication is not utilized by these two methods, some of hardware resources are in a sense wasted.

Method four is superior to the other methods in all but two areas – at the speed of switchover and at compatibility to the third party hardware.

Method one appears to be a solid method that can support telecom level protection. The only problem with method one is that software upgrades may cause problems with it, if functionality of the routing daemons differ between software versions.

The selected solution is to implement both method one and method four. Methods one and four can co-exist on same TCP connection and therefore trying to negotiate method four never causes any harm. Should the remote router need the method four services to recover from a crash, they are available. Method four can also be used to update incompatible software versions. Method one can be used in combination with method four, if the remote peer does not support method four or if the service level agreement requires faster switchovers than method four can offer.

Together methods one and four can offer unparalleled performance and compatibility. Combining these two methods virtually removes all the defects of these two methods. The inconvenience with this approach is that implementation effort is quite large, as both methods should be implemented.

Chapter 13

Adaptation to the carrier class router environment

BGP-4 routing daemon is a part of a off-the-self routing toolkit available from a third party vendor. This daemon will be extended to support the graceful restart mechanism. By default all connections shall be negotiated with the graceful restart capability.

As our routers are end-to-end managed, networks often contain continuous segments of our devices. Therefore graceful restart extension is guaranteed to be supported in most BGP-4 connection. Interfaces to third party devices are at the network edge and at the network core edge. It is likely that third party routers are only encountered at these edges. This reduces the risk caused by the possibility that the graceful restart mechanism for BGP-4 is not widely adopted – it can still be used between our equipments.

When a remote router does not negotiate the graceful restart capability, the underlying TCP connection is protected using method 1. As method 1 was specifically designed for the carrier class router environment, it requires no specific adaptation. TCP connection can also be protected if network manager decides that switchover times offered by the graceful restart extension for BGP-4 are insufficient.

The use of method 1 requires that protected data structures are compatible. If software with incompatible data structures is updated to either control card, method 1 protection is lost from all protected connections. Connections themselves would be interrupted during the next switchover. To minimize interruption, the routing software always tries to negotiate the graceful restart extension, regardless of the used of method 1 protection.

Purpose of method 1 protection is to offer protection for connections to legacy devices that do not support the graceful restart mechanism. Method 1 is also used for most important connections to provide enhanced switchover performance. The graceful restart mechanism is used to provide lightweight protection and transparent software update capabilities.

Chapter 14

Conclusions

This work investigated several possibilities for protecting BGP-4 routing connections. Early chapters of this thesis introduced the carrier class router architecture and protocols and interfaces related to this work.

Chapter 6 presented current solutions for high availability routing. It was observed that none of the introduced methods offered level of availability required by the carrier class router.

Chapter 7 introduced the concept of protection. The benefits of the protection were noted and commonly used implementation strategies were discussed. It was observed that protection is hard to implement on IP based devices. Chapter 8 introduced previous work on area of protecting routing connections. Existing methods based on protecting TCP connections appeared to be lacking in one or more ways, whereas extensions to routing protocols appeared to be quite promising.

Chapter 9 set forth requirements that were used for evaluating different protection mechanisms. The requirements were used to derive the minimum set of data that must be protected to ensure successful fulfillment of the requirements.

Chapter 10 proposed two methods for performing replication necessary for many protection methods. It was observed that MMU assisted DMA transfer is superior to traditional serialization based bulk transfer approach.

Chapter 11 proposed several methods for protecting TCP based routing connections. Methods were described in detail and their good and bad sides were discussed. It became apparent that although several methods are more or less suitable, implementing most of them is difficult.

In chapter 12 the requirements defined in chapter 9 were used to compare proposed protection methods. None of the proposed methods alone could fulfill all the requirements. Combination of protecting TCP state machines with logical synchronization (method 1) and of graceful restart mechanism for BGP-4 (method 4) was chosen as the protection method for BGP-4 routing connections. This combination does fulfill all the requirements. Adaptation of these methods for the carrier class router environment was described in chapter 13.

On the chosen solution a new BGP-4 capability is negotiated with remote peers. This capability allows re-establishment of broken TCP connections. When such capability can not be negotiated, or

better response time is necessary, TCP state machines can be executed on both control cards. To avoid state machine divergence, TCP events on both control cards are performed on the same order.

To summarize, this thesis has introduced a method for protecting TCP state machines and combined it with a relatively new BGP-4 extension to allow protection of BGP-4 routing connections. If the graceful restart mechanism becomes widely implemented, the need for protecting underlying TCP connections will diminish.

Bibliography

- [1] L. Alvisi, T. Bressoud, A. El-Khashab, K. Marzullo, and D. Zagorodnov. Wrapping server-side TCP to mask connection failures. *Proceedings of Infocom 2001*, pages 329–328, 2001.
- [2] T. Bates, R. Chandrasekeran, and E. Chen. RFC 2796: BGP route reflection – an alternative to full mesh IBGP, April 2000. Status: PROPOSED STANDARD.
- [3] S. Bellovin. RFC 1948: Defending against sequence number attacks, May 1996. Status: INFORMATIONAL.
- [4] R. T. Braden. RFC 1122: Requirements for Internet hosts — communication layers, October 1989. Status: STANDARD.
- [5] R. Chandra and J. Scudder. RFC 2842: Capabilities advertisement with BGP-4, May 2000. Status: PROPOSED STANDARD.
- [6] R. Chandra, Y.Rekhter, R. Fernando, J. Scudder, and E. Chen. draft-ietf-idr-restart-00.txt: Graceful restart mechanism for BGP, December 2000. Work in progress.
- [7] D. Comer. *Internetworking with TCP/IP, Volume 1: Principles, protocols and architecture*. Prentice Hall, 1995.
- [8] A. Farrel, P. Brittain, P. Mathews, and E. Gray. draft-ietf-mpls-ldp-ft-01.txt: Fault tolerance for LDP and CR-LDP, February 2001. Work in progress.
- [9] C. Fetzer and S. Mishra. Transparent TCP/IP based replication. *Proceedings of the 29th IEEE International Symposium on Fault-tolerant Computing*, June 1999.
- [10] A. Heffernan. RFC 2385: Protection of BGP sessions via the TCP MD5 signature option, August 1998. Status: PROPOSED STANDARD.
- [11] J. Heinanen. Tellabs ATM, February 2000. Teleware course material.
- [12] J. Steward III. *BGP4, Inter-Domain Routing in the Internet*. Addison-Wesley, 1999.
- [13] V. Jones. *High Availability Networking with Cisco*. Addison-Wesley, December 2000.

- [14] Phil Karn and Craig Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM Transactions on Computer Systems*, 9(4):364–373, November 1991.
- [15] S. Knight, D. Weaver, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand, and A. Lindem. RFC 2338: Virtual router redundancy protocol, April 1998. Status: PROPOSED STANDARD.
- [16] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. RFC 1928: SOCKS protocol version 5, April 1996. Status: PROPOSED STANDARD.
- [17] T. Li, B. Cole, P. Morton, and D. Li. RFC 2281: Cisco Hot Standby Router Protocol (HSRP), March 1998. Status: INFORMATIONAL.
- [18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP selective acknowledgment options, October 1996. Status: PROPOSED STANDARD.
- [19] D. Mills. RFC 1305: Network time protocol (version 3) specification, implementation, March 1992. Status: DRAFT STANDARD.
- [20] J. Moy. *OSPF Anatomy of an Internet Routing Protocol*. Addison-Wesley, May 1999.
- [21] J. Nagle. RFC 896: Congestion control in IP/TCP internetworks, January 1984. Status: UNKNOWN.
- [22] J. Postel. RFC 793: Transmission control protocol, September 1981. Status: STANDARD.
- [23] Y. Rekhter and P. Gross. RFC 1772: Application of the Border Gateway Protocol in the Internet, March 1995. Status: DRAFT STANDARD.
- [24] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4), March 1995. Status: DRAFT STANDARD.
- [25] R. Stevens and G. Wright. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1994.
- [26] C. Villamizar, R. Chandra, and R. Govindan. RFC 2439: BGP route flap damping, November 1998. Status: PROPOSED STANDARD.