The Design of Parallel Queue Processing Strategy in Reflective NetGAP

Songsen Yu¹, Yun Peng²*, Biqing Zeng¹, and Jian Yan³

¹Department of Computer, South China Normal University, Foshan 528225, China
E-mail: yss8109@163.com

²College of Computer Information Engineering, Jiangxi Normal University, Nanchang 330022, China

³Faculty of Automation, Guangdong University of Technology, Guangzhou 510006, China

Abstract—The Reflective GAP is hardware-based and its purpose is to rapidly mirror buffers between the Non-Trusted memory and the Trusted memory. Its activities are achieved using store & forward of memory blocks. Its software system comprises with seven main basic modules. Between these modules we adopt a concept of “worker with queue”, the overall system is divided into X parallel processing paths. Each path consists of some workers consistent with these modules. Each worker pops messages from his queue, and dispatches the message to the relevant session module. We carefully design its message streaming scenario in different situation. The Reflective GAP can keep good security and implement high Real-time data transmission.

Index Terms—NetGAP, Reflective GAP, parallel processing, message streaming

I. PREFACE

Although the technique of traditional Network Isolation card ensures network security, it forms an "isolated island" owing to lack of information exchange mechanism and at the same time limits the development of its application. In the recent years, the rapidly-developed NetGAP technique is on the basis of physical isolation. NetGAP can physically isolate the internal and external networks. While NetGAP technique ensures the security, it not only solves the difficulty of information exchange between networks, but also breaks through the application difficulty caused by security [1,2].

NetGAP always constitutes with two parts: the trusted side and the non-trusted side. The trusted side is the side that is connected to the trusted network, while the non-trusted side is the side that is connected to the non-trusted network. It is implemented in both hardware and software.

There are seven main software modules that comprise the “GAP” system. Six out of the seven modules are actually twin modules that reside on both sides of the “GAP”, the trusted and non-trusted sides. The trusted side (“Trusted”) is the side that is connected to the trusted network, while the non-trusted side (“Non-Trusted”) is the side that is connected to the non-trusted network. The remained module (the one that does not have a twin module) is the security module that handles the protocol security decisions in the system. For security reasons, it resides, on the trusted side of the “GAP”.

A single datagram goes through five modules in its life cycle in the system. The Fig.1 shows the basic seven modules and the flow of a single datagram.

A datagram can come from both sides of the network (the trusted and non-trusted sides). In both cases, it passes the same route of modules. The only difference between one route to the other, is where the datagram crosses the “GAP” chip and moves from one CPU to the second CPU.

The route of a datagram starts at the “TCP module” which is a proxy module. It stops the datagram, packs the data buffer to an application message and passes the message to the “Parser Modules”. The “Parser Module” does the protocol parsing analysis, creates a proprietary “parsed message” and sends it to the “Trusted Module”. The “Trusted Module” does most of the security decisions and transfers the “approved” parsed messages to the “Regenerator Module”. Here the “Parsed” data is regenerated to a similar but not equal to the original “buffered form” message. Finally, it is handled by the seconds “TCP Module” proxy, which mirrors the original session[3].

II. “WORKER WITH A QUEUE”

The concept of “worker with queue” is used to describe how modules are being called and operate. Basically the overall system is divided into X parallel processing paths. Each path consists of TCP/IP worker, Parser worker, Session Manager worker, Protocol regenerator worker and the other side TCP/IP worker. Each worker pops messages from his queue, and...
dispatches the message to the relevant session module. The concept can be explained by figure 2.

### III. MESSAGE TYPES

TCP/IP messages: Used between TCP/IP module and the Parser module[4,5].

- ConnectSocket – used for request/replay of establishing new connection through the system. (only socket to socket connection).
- CloseSocket – used for request/replay for both closing the socket and the session. Instruct managers to free all allocated resources for this session.
- Flow control – Acknowledges messages after successful process completion.
- EnableService – used for instructing the TCP/IP module to open a listen socket.
- Data message – raw data message read from the socket.

### IV. SCENARIOS

Under the concept of “worker with queue” we will describe how do the main modules behave during a regular session. Different scenarios are outlined as follows.

#### A. “Http Streaming” open connection successfully

The Object creation is done only when first TCP packet arrives. The scenario is designed that the “TCP module” firstly sends connect message through default queue to the “Parser Module”. It forwards the message to Session Manager. The Session Manager then Call Rulebase to approve connection. If it is accepted Unique Session ID will be generated otherwise it will send close with an application message. After the “TCP module” receives the connecting message, it does asynchronous connection. If the TCP/IP Worker does not get an acknowledgement, it sends a close with an application message. The entire scenario is implemented as shown by the figure 3.

#### B. “Http Streaming” connection to remote host failed

The “TCP module” firstly generates Unique Session ID and assigns new session. The “Parser Module” creates parser for new session ID only when first packet arrives, otherwise it does nothing. The Session Manager will call “Connection Approve agent” to ask Rulebase to approve connection. If accepted it will send connecting message, otherwise it will send close with an application message. The “TCP module” does asynchronous connection. If the
connection Fails, The “TCP module” deletes all allocated resources for session ID and returns failed connection message.

C. “Http Streaming” connection rejected by rulebase

The “Http Streaming” Connection rejected by rulebase is designed as the figure 4.

D. “Http Streaming” regular session data flow

The “TCP module” packs TCP packet to message, marks OpenObject flag and sends it to next module. The “Parser Module” creates parser for new session ID. The “ Trusted Module” creates session Manager. Session Manager will ask rulebase, update stateMachine and send application message to the “Regenerator Module”. The “Regenerator Module” will convert application message to TCP packet and send it to the “TCP module”. The following figure 5 explains the concept.

E. “Http Streaming” regular session close

When the “TCP module” receives Close socket, it will mark socket status as closed and forward it. The “Parser Module” will delete parser. Session Manager will ask rulebase and update stateMachine. The “Regenerator Module” will delete regenerator. The “TCP module” then frees session resources and returns Close socket Ack.

V. CONCLUSION

In above sections, we introduced our parallel queue processing Strategy between NetGAP modules based on “worker with queue”. Under this Strategy message can be safely transferred between modules, its message streaming scenario in different situation can be designed easily. The Reflective GAP can keep good security and implement high Real-time data transmission.

ACKNOWLEDGMENT

It is a project supported by The National Natural Science Foundation of China (No. 40774005) and Guangdong Natural Science Foundation (06023131).

REFERENCES