The ARTCP header structure, computation and processing in the network subsystem of Linux kernel.

Anatoliy Sivov
Yaroslavl State University
Yaroslavl, Russia
mm05@mail.ru

V.A. Sokolov (research supervisor)
Yaroslavl State University
Yaroslavl, Russia

Abstract—**ARTCP is a transport level communication protocol based on TCP. It uses temporal characteristics of data flow to control it, that allows to split algorithms of congestion avoidance and reliable delivery. The article discusses possible ARTCP header structure and practical aspects of forming the header and calculation of the header fields. It demonstrates the possibility of transparent replacement of TCP with ARTCP due to flexible ARTCP connection setup implementation and ARTCP packets structure compatibility with TCP. The questions of precise time dispatching of the received packets are discussed. The Linux kernel interfaces for time measurement are described as well as the clock source abstraction layer and its implementation.**

Networking; transport protocol; ARTCP; time measurement; Linux kernel.

I. Introduction

TCP is the most widespread transport protocol with the reliable data delivery today. It has become the industry standard de facto. However, this protocol is not ideal. It has at least two big disadvantages. The first of them is that its data flow management algorithm results in periodic network congestion by design. It leads to unnecessary packet loss and latency increase. The second is inefficient bandwidth use in the unreliable physical environment (i.e. wireless networks) where BER can be reasonably high. This disadvantage comes from TCP impossibility to distinguish packet loss due to congestion and packet loss due to some transmission errors.

Adaptive Rate TCP (ARTCP) is a transport protocol with the reliable data delivery that uses some TCP principles but tends to solve these two TCP's disadvantages. ARTCP uses temporal characteristics of data flow in its data flow management algorithm. It allows ARTCP to determine bandwidth efficiently without need in periodic network congestion. The main feature of ARTCP is a logical separation of error correction and data flow management. Due to this separation ARTCP is able to use bandwidth more efficiently than TCP in unreliable environment [1]. The ideas of ARTCP and protocol's description can be found in [1] and [2].

This article is a research-in-progress report about ARTCP implementation in Linux kernel. It considers the following things: transparent replacement of TCP with ARTCP, coexistence of TCP and ARTCP, ARTCP header structure and processing, and computation of ARTCP header fields values in Linux kernel.

The Linux kernel is chosen as the target platform of ARTCP implementation for several reasons:

- Linux is an open source system. It allows to use its source code and modify it. So, Linux allows the most flexible implementation process it could be what is very important while implementing protocol that is closely associated with the existing one. It makes implementation to be much more easy and efficient comparing with writing the module for operating system with closed sources.
- Linux has very good network subsystem. When implementing transport protocol it is very desirable to have good implementation of underlying protocols as well as networking implementation at whole. Linux has advantages of very good networking stack with well-implemented layers abstraction and object-oriented socket concept. The other advantage is a TCP implementation modularity.
- Linux is a popular, industry-choice operating system. Linux is the most popular operating system in the industry comparing with the other open source operating systems. The wish to create an implementation of ARTCP for OS widely used in the industry influenced on the choice among variety of open source operating systems.
- Linux can run on various hardware platforms. Linux has support of many hardware platforms including x86, ia64, x64, arm, avr, mips, ppc and so on. ARTCP implementation written hardware-independently will be supported on all of these platforms.

II. ARTCP and TCP

It is very important to have TCP working on the system which supports ARTCP. Taking into account that ARTCP is considered as transparent replacement of TCP the responsibility for this lies on ARTCP implementation. It is so because applications (or overlying protocols such as HTTP) does not know whether they establish TCP or ARTCP connection unlike the situation when they use the other transport protocols (UDP or something more exotic like SCTP).

We have chosen to support TCP on ARTCP-featured systems in two ways. At first, operating system administrator must have a simple capability to choose whether to use TCP or ARTCP. This problem is solved with addition of **tcp_enable_artcp** kernel parameter with possible values 0 or 1 where 0 means that ARTCP is disabled (and TCP is used) and 1 means that ARTCP is enabled (and ARTCP is used if possible for all connections). Like any other kernel parameter this parameter is accessible in runtime with /proc/sys/ interface. Its value can be changed with writing 0 or 1 to **/proc/sys/net/ipv4/tcp_enable_artcp** file. Also its value can be set in /etc/sysctl.conf file in the same way as any other kernel parameter.

Secondly, ARTCP implementation must be able to fall back to TCP if the other end of connection does not support ARTCP. This capability allows to have ARTCP enabled in the TCP world. It is very useful to be able to switch to TCP in the kernel without any packet retransmission or disturbing overlying protocol or an application that uses ARTCP/TCP connection but imposes ARTCP packet structure and connection setup to be TCP-compatible. This article suggests ARTCP header structure and connection setup that are TCP-compatible and makes it possible to fallback to TCP at any time of connection. Also it represents implementation of ARTCP header processing in Linux networking subsystem and discusses the questions of ARTCP header fields values calculation.

III. ARTCP HEADER STRUCTURE

As mentioned above ARTCP header structure must be TCP-compatible. Let us consider TCP header and how it is possible to extend it to fit ARTCP needs.

TABLE I. TCP HEADER

Bit	0-3	4-7	8-15	16-31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Data offset	Reserved	Flags	Window size
128	Checksum			Urgent data pointer
160	Options (optional field)			

TCP has a native support to extend header called TCP options. TCP options are considered in [3] and succeeding RFCs and have the following format: first byte contains option number, second byte contains option length (in bytes including 2 bytes for number and length fields) and 0 or more bytes (specified in the length field) contains option value. There are only two exception for this format. Option number 0 is one byte long. It is used to mark the end of options list. The other exception is an option number 1 which is used for padding to align other options on 32-bit boundaries. TCP header allows up to 40 bytes to be used for options list.

ARTCP requires only two extra fields for its functionality: PS field and TI field. Each one of them can be represented as 32 bit number. PS (Packet Sequence) field holds unique packet sequence number (modulo 2^{32}, of course). According to [1] this field must be presented in every ARTCP packet with payload data. ARTCP receiver uses this field to determine whether the packet received is the next packet in the stream comparing with the previous received packet. So, the value of PS field in ARTCP is to help receiver to distinguish the packet in order sent first time from the packet in order sent again (due to some packet loss). Indeed data presented in TCP header are enough to distinguish segments in order from segments out of order using the sequence number field. However, in terms of TCP there is no difference between the segment sent first time and the segment sent again (due to retransmission) because both of them share the same sequence number and there is no any indication of retransmitted segment in TCP.

The second field, TI (Time Interval) is used by ARTCP to compute the value of stream's duty ratio. The paper [4] suggests to carry the time interval measured between two consecutive moments of ARTCP packets arrival. It is considered more useful to carry time intervals in the TI field instead of carrying duty factor what was suggested in [1]. ARTCP receiver puts this field in every its acknowledgment packet (packet that has ACK flag). It is necessary to use real (“human”) time units for time interval resolution in TI that must not depend on hardware used by sender or receiver (i.e. these units must not be CPU ticks or something like that). The paper [4] suggests to use microseconds for this purpose. Time measurement with this resolution is possible on the most of hardware used nowadays and has a sufficient precision for the existing problem. TI field may be represented with 32 bit number.

Both TI and PS can take the form of TCP options in ARTCP header. In this case they must take at least 6 bytes (8 bytes to keep the 32 bit alignment) – 1 byte for option number, 1 byte for option length and 4 bytes for option value (and 2 bytes for alignment). For today implementation PS field may use option number 253 and TI field may use option number 254. These option numbers are chosen in conformity with RFC 4727 [5] to use in experiments. They must be changed later in conformity with RFC 2780 [6]. The use of TCP options numbers 253 and 254 is regulated in RFC 3692 [7].

Summing up, ARTCP header is a valid TCP header extended with two TCP options called PS and TI. Every ARTCP packet with payload data as well as packet with SYN or FIN flag contains PS field with packet sequence number in the header. This number is one more than the number of previous segment transmitted by the sender (excluding case of retransmission during connection setup). Namely, if the segment with value N in the field PS was transmitted by sender but was not delivered (or its acknowledgment was not received by sender) then this segment is retransmitted (with possible repacketization) but has value N+1 (modulo 2^{32}) in the PS field. Every ARTCP packet with ACK flag must contain TI field in its header. TI field must have 0 as value if this packet is an acknowledgment for a packet that contains in PS field value that differs from the value of previous received packet incremented by one modulo 2^{32}. Otherwise, field TI must contain calculated value of the time interval. Both fields PS and TI are written in the network order.

IV. ARTCP header processing

ARTCP shares a lot of algorithms with TCP, also ARTCP implementation must allow fallback to TCP if one of connection ends does not support ARTCP. So it's decided to use existing network subsystem of Linux kernel, implementation of Ipv4/TCP stack in particular, to implement ARTCP.

As described above, ARTCP header differs from TCP header with presence of PS and TI fields only. These fields are represented in the form of TCP options so that it is necessary to modify the code that implements TCP options reading and writing to implement reading and writing of ARTCP header.

To form TCP options to be sent in the header Linux kernel uses **struct tcp_out_options**, the structure that contains the fields with values of different TCP options supported by network stack of Linux and bit field **options** to set flags that indicates which options must be written to the header of the current TCP packet. To implement writing of ARTCP header the fields for TI and PS values were added to this structure, also bit flags, that indicates the presence of the fields and are used in options bit field, were created.

For this structure to be filled correctly the functions that initializes the instance of this structure were modified. These functions are **tcp_syn_options**, **tcp_synack_options** and **tcp_established_options.** They forms options for SYN packets, SYN-ACK packets and the other packets, respectively. The modified functions checks whether socket is in the ARTCP mode and if so adds information about needed PS or/and TI field.

To write TCP options into network buffer that contains the header **tcp_options_write** function is used. This function is modified as well to write PS and TI fields to the header if it is specified in the instance of the modified **struct tcp_out_options**. All these modifications make it possible to form ARTCP packets in the Linux network subsystem.

To parse the options of the TCP header in the packet received Linux calls **tcp_parse_options** function, which analyzes the received data and forms the instance of **struct tcp_options_received** by writing received values into it. To support ARTCP this structure was extended with fields **ps** and **ti** that contains values of fields PS and TI of the received ARTCP packet and bit field **artcp_options** that determines which ARTCP fields were actually presented in the header (PS, TI, both or none). Also **tcp_parse_options** function must be modified to handle ARTCP fields and form the modified structure.

To handle ARTCP packets properly it is necessary to process received ARTCP fields depending on the connection state and the presence/absence of payload data in the received packet. In IPv4/TCP stack received SYN packet is processed in **tcp_v4_conn_request** function. The modified code of this function checks the kernel parameters set by administrator by reading **sysctl_tcp_enable_artcp** variable which has information whether ARTCP is globally enabled in the system or not. If ARTCP is enabled **tcp_v4_conn_request** checks the presence of field PS in the received SYN packet and the correctness of its value as well as the absence of field TI. If all checks are passed then function puts socket in ARTCP mode and initializes all resources needed by ARTCP connection. Otherwise, the function puts socket in TCP mode.

If the socket has already sent SYN packet (and is in SYN-SENT state) then the packets received with this socket are handled with **tcp_rcv_synsent_state_process**. The modified code of this function checks the presence of PS and TI fields in the header and correctness of their values when SYN-ACK packet is received for socket in ARTCP mode. If checks are not passed then the function puts the socket in TCP mode. Otherwise, function initializes all resources needed by ARTCP connection.

For established ARTCP connection all received packets are handled with **tcp_rcv_established** function. The modified code of this function processes ARTCP packets for the socket in ARTCP mode. If the header of the received packet has no needed fields PS (for packet that has payload data) or TI (for packet that has ACK) or the header of the packet without payload data has field PS then socket falls back to TCP. If ACK packet received then the value of field TI is passed to data flow management algorithm of ARTCP. For packet that has payload data the function checks the value of field PS. If this packet is the next (after previous received packet) packet sent by the other end according to this field then it is necessary to calculate the difference between the time of arrival of these two packets to send it in the field TI of the acknowledgment packet. Otherwise, the ACK packet will contain 0 in the field TI.

V. Time measurement for TI field in Linux

TI requires time measurement with microseconds resolution what may be nontrivial problem. Linux kernel guarantees the availability of so-called “system clock” represented with **jiffies** interface. **Jiffies** can be considered as read only global variable which is updated with **HZ** frequency. **HZ** is a compile-time kernel parameter whose reasonable range is from 100 to 1000 Hz [8]. So, it is guaranteed to have an interface for time measurement with 1-10 milliseconds resolution.

The availability of more precise techniques for time intervals measurement is hardware-dependent. Let us consider x86 architecture as an instance. All IMB-compatible PCs have Programmable Interval Timer (PIT) known as chip Intel 8523 (or Intel 8524 and other analogues). This chip (or an analogue, i.e. south bridge of the motherboard may have this functionality) has three independent 16-bit counters called channels. Channel 0 usually is used for clock interrupts generation. Channel 1 assists in generating timing for DRAM memory refreshes. And channel 2 commonly generates PC speaker tones. PIT allows to achieve 1 ms time resolution.

The other clock source is Real Time Clock (RTC). RTCs usually have an alternate source of power, so they can continue to keep time while the primary source of power is off or unavailable. RTC's functionality is provided with south bridge in the modern motherboards. However, RTC allows time measurement with 1 ms resolution.

The most modern x86 motherboards have Advanced Programmable Interrupt Controller (APIC) and APIC timer as a consequence. This timer's frequency equals CPU bus frequency what allows time measurement with a high resolution (about 10 nanoseconds). The other benefit is that in contradistinction to PIT and RTC local APIC timer does not require call to I/O port. The most uniprocessor PCs above Pentium 4 explicitly prohibits APIC by disabling it in BIOS.

Also systems, that have a support of Advanced Configuration and Power Interface (ACPI), have so-called Power Management timer (PM timer). Unlike APIC timer, it is possible with PM timer to have a reliable time independently on CPU speed changes due to active power management with OS.

At the beginning of 2000s Intel and Microsoft corporations has developed High Precision Event Timer (HPET) [9]. This timer has a high frequency (not less than 10 MHz) and uses 64-bit counter. Often it is a most preferable high-precision clock source in the system.

In addition to peripheral timers x86 computers have on-chip (on-CPU) 64-bit counter called Time Stamp Counter (TSC). Comparing with the other counters this one has advantages of less read latency and high resolution. The frequency of this counter on different CPUs varies and can equal CPU frequency or CPU bus frequency. There are two major problems to use this counter as clock source. The first one is that Time Stamp Counters may be not synchronized between cores of SMP [10]. The second is that frequency of TSC may be non-constant (due to power management or processor frequency changes on idle and so on).

Intel's software developer's manual [11] describes in depth the differences in TSC implementation on different Intel CPU families. It also describes the way to recognize whether CPU has TSC with invariant rate. Most AMD processors have TSC that is unusable as a reliable clock source because of certain circumstances.

The facilities for time interval measurement in x86 architecture listed above give an idea of the difficulty to solve this problem more precisely for different processors. The support of other hardware architectures (arm, mips and so on) highly increases this difficulty. The other problem is a non-triviality of time units translation from “machine” time units used in the chosen device to “human” time units (for example, microseconds needed by ARTCP). Reading the report [12] shown in 2005 in Ottawa (Canada) at a symposium devoted to Linux you can get an idea on the complexities associated with the solution of this problem.

Fortunately, Linux kernel provides the means for solving these problems. To have a possibility to use different hardware counters and timers “clock source” concept is implemented in Linux kernel. According to this concept, each hardware architecture supported by Linux implements for each available facility a “clock source” interface. It does it by initializing an instance of **struct clocksource** interface and registering it in operating system with call to either **clocksoure_register_khz** or **clocksource_register_hz**. **Struct clocksource** has field **rating** that allows Linux to choose the best “clock source” available for the specified hardware. Best “clock source” corresponds to the registered instance of **struct clocksource** with the biggest value of field **rating**. The values of this field are logically interpreted in that way: 1-99 – unfit for real use (only available for bootup and testing purposes), 100-199 – base level usability (functional for real use, but not desired), 200-299 – good (a correct and usable clocksource), 300-399 – desired (a reasonably fast and accurate clocksource), 400-499 – perfect (the ideal clocksource, that is a must-use where available).

Since the best “clock source” has been chosen Linux kernel is able to read its counter values by calling the function passed in field **read** of **struct clocksource**. This function returns the value in abstract “machine” time units represented with **cycle_t** data type. Linux kernel can use the values of field **mult** and field **shift** of **struct clocksource** to translate this value to nanoseconds.

Linux kernel provides various interfaces for indirect work with “clock sources” and to retrieve the values of time in “human” units. The most interesting of them are **getnstimeofday** and **getrawmonotonic** functions. Both of these functions return the value of time as an instance of **struct timespec**. This structure consists of two fields: **tv_sec** that carries seconds and **tv_nsec** that carries nanoseconds. The most significant difference between these functions is that the former unlike the second uses NTP correction to adjust value it returns. The absence of this correction in **getrawmonotonic** allows to use this function to compute time intervals where the correspondence of the time set on the machine to the actual time does not matter. Based on these considerations, ARTCP implementation uses **getrawmonotonic** interface to calculate time intervals between two consecutive received ARTCP packets with payload data.

The first time reading with **getrawmonotonic** happens in **artcp_init** function in the process of ARTCP connection initialization when either socket, that sent ARTCP packet with ACK, receives ARTCP packet with SYN-ACK, or socket receives ARTCP packet with SYN (and ARTCP is globally enabled). Subsequent readings occur when ARTCP packets with payload data are received. Moreover, if the value of the field PS in the received packet is one greater (modulo 2^{32}), than the value of the field PS in the previous received packet, then the value for the field TI of the acknowledgment packet is calculated with call to **artcp_ts_diff_to_ti** function. The function **artcp_ts_diff_to_ti** takes two **struct timespec** arguments, that represent the time interval, and returns the difference between these moments of time in microseconds.

Summarizing the material described in this article, we can conclude that, having made the above changes in the source code of Linux network subsystem, we get the implementation of ARTCP packets processing (receiving, sending, calculation of field values) that is completely independent from the implementation details of data flow management function and the other parts of ARTCP. Moreover, the implementation is cross-platform (in the hardware) and uses the best available hardware facility for time intervals calculation with the ability to increase the resolution of the field TI up to nanoseconds. It is also worth nothing that the implementation does not conflict with the existing functionality of Linux network subsystem, allowing the latter to use TCP connections simultaneously with ARTCP and even switch ARTCP connections to TCP mode.

[1] I. V. Alekseev, V. A. Sokolov, D.U. Chaly. Modeling and analysis of Transport protocols for computer networks. Yaroslavl State University, 2004. (in Russian)

[2] I. V. Alekseev, V. A. Sokolov Compensation Mechanism for Adaptive Rate TCP. // 1-St International IEEE/Popov Seminar "Internet: Technologies A and Services". P. 68-75, October 1999

[3] J. Postel. Transmission Control Protocol. // RFC 793 (STD7). 1981.

[4] I. V. Alekseev, S. A. Merkulov, A. A. Sivov. “Aspects of practical implementation of ARTCP in Linux kernel 2.6” // Modeling and analysis of information systems. Volume 17, №2. Yaroslavl: Yaroslavl state university, 2010. P. 144-149 (in Russian)

[5] B. Fenner. Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers. // RFC 4727. 2006.

[6] S. Bradner, V. Paxson. IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers. // RFC 2780. 2000.

[7] T. Narten. Assigning Experimental and Testing Numbers Considered Useful. // RFC 3692. 2004.

[8] J. Corbet, A. Rubini, G. Kroah-Hartman. Linux Device Drivers. 3rd edition. O'Reilly, 2005.

[9] IA-PC HPET (High Precision Event Timers) Specification. Rev. 1.0a. Intel Corporation, 2004.

[10] AMD Technical Bulletin – TSC Dual-Core Issue & Utility Fix. Advanced Micro Devices, Inc. 2007.

[11] Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3A: System Programming Guide, Part 1. Intel Corporation. January 2011.

[12] J. Stultz, N. Aravamudan, D. Hart. We Are Not Getting Any Younger: A New Approach to Time and Timers. // Proceedings of the Linux Symposium. Vol. 1. P. 219-232, July 2005.