FRTP: Fixed Rate Transport Protocol

-- A modified version of SABUL for end-to-end circuits

Xuan Zheng, Anant Padmanath Mudambi, and Malathi Veeraraghavan
University of Virginia, 351 McCormick Rd, Charlottesville, VA 22904

Abstract[1] -- New applications requiring high and steady bandwidth have been introduced in recent years. Wide area circuit-switched networks appear to be a good choice for many of these applications but not much work has been done on designing a transport protocol for such networks. In this paper we identify the features that a transport protocol for circuit-switched networks should have. We then modify the SABUL protocol and implementation so it has these properties. Finally, through a series of experiments we examine the performance of this "Fixed Rate Transport Protocol" and demonstrate its suitability for end-to-end circuits.

I. INTRODUCTION

A need for high-speed networks that can offer deterministic bandwidth and latency has been identified for e-Science projects that involve geographically distributed scientists [1][2]. Despite the recent advances in middleware and grid technologies [3], requirements of several large-scale e-Science applications are not met by current networks.

A number of next generation IP-based network testbeds, such as Abilene [4], ESnet [5], NSF Teragrid [6], have been deployed to meet this need. These testbeds provide high-bandwidth connectivity between the participant nodes, and hence are good platforms to support the needs of new e-Science applications. Since TCP has been shown to be inefficient in High-Delay-Bandwidth-Product (HDBP) environments [7] a number of TCP enhancement have been proposed to upgrade TCP's congestion control and/or flow control. Examples include HighSpeed TCP [8], FAST TCP [9], and ScalableTCP [10]. These protocols are proposed to be implemented in the kernel space and require modifications to the operating system. To avoid the complexity of kernel-level change, other groups of researchers have proposed new transport protocols which are implemented as application-level processes running on top of UDP. Examples include SABUL [12], UDT [13], Tsunami [14], and RBUDP [15]. These protocols use rate-based congestion control instead of TCP-like window-based control because rate-based flow control is regarded as a more efficient solution for high-speed networks [11].

Meanwhile, the circuit-switched community is deploying a number of optical circuit-oriented testbeds, e.g., CANARIE's CA*net 4 [16], OMNInet [17], SURFnet [18], and UKLight [19]. These networks allow for end-to-end circuits to be provisioned for the dedicated use of e-Science applications. Some of these networks use all-optical switches with the granularity of a circuit being a single wavelength, while others use hybrid electronic/optical switches that provide sub-lambda granularity.

Along the same lines, we proposed an end-to-end optical networking solution called Circuit-switched High-speed End-to-End Transport ArcHitecture (CHEETAH) [20]. In this solution, we proposed using end-to-end Ethernet/Ethernet over SONET circuits provisioned dynamically just for the duration of a single file transfer. CHEETAH is designed to be an add-on to the primary Internet service. A routing decision is made before each transfer to decide whether or not to set up an end-to-end circuit. If the answer is negative or the circuit setup fails, the application falls back on the primary Internet path.

A number of problems need to be solved before scientists can start enjoying the end-to-end connectivity offered by CHEETAH or other circuit-switched solutions in a seamless fashion. One of these is to design a transport protocol suitable for data transfers on high-speed end-to-end circuits. This is the focus of

1.This work was carried out under the sponsorship of NSF grants, ITR-0312376 and AIN-0335190 at UVA.

this paper.

The rest of the paper is organized as follows. In Section II we look at past work and motivate the need for a new transport protocol for circuit-switched networks. Section III describes our design rationale for the new transport protocol while, in Section IV we present its implementation details. Section V presents the results of some experiments carried out in our local testbed. We conclude the paper in Section VI.

II. MOTIVATION

None of the high-speed transport protocols, the TCP enhancements or the UDP based protocols, mentioned in the previous section, except RBUDP, are directly suited for end-to-end circuits. This is because these protocols were developed for packet-switched networks. In packet-switched networks, transport protocols are designed to run congestion control mechanisms to adjust sending rates during the data transfer based on network congestion levels. In circuit-switched networks, since resources are reserved in a dedicated manner for circuits, congestion is handled during circuit setup. Once the circuit is provisioned successfully, **congestion control** functionality is **not** required during the data transfer. This means that there is no congestion related reason to adjust the sending rate. On the contrary, the main objective of a transport protocol for an end-to-end circuit should be to maintain the sending rate constant at a value equal to the circuit rate. If the sending rate falls below the circuit rate, the circuit will be underutilized (circuit utilization can be defined as the proportion of reserved circuit capacity actually used for data transfer during the time for which the circuit is held open). If, on the other hand, the sending rate exceeds the circuit rate, there will be data loss. Both conditions should be avoided, which means the sending rate should be held constant at the circuit rate.

As stated earlier, RBUDP is the only current solution that is specifically targeted at circuit-switched networks and implemented with rate-based flow control [21]. In RBUDP, the sender transmits the whole file at a fixed rate to keep the circuit fully utilized during the data transfer. However, the sender retransmits all lost packets at the end of user-data transfer after waiting for and receiving a notification of lost packets from the receiver. This will impact circuit utilization because the circuit lies idle after the completion of the initial transfer while waiting for the notification of lost packets.

Thus, given that high-speed transport protocols developed for packet-switched networks cannot be used directly for high-speed end-to-end circuits and the utilization issue with RBUDP, we decided to create a transport protocol specifically for file transfers on end-to-end circuits. Drawing attention to the need for maintaining a fixed rate at the sender, we call our protocol Fixed Rate Transport Protocol (FRTP).

After we designed FRTP we decided that instead of starting our implementation from scratch, we could take advantage of the implementation of existing high-speed protocols and modify one of them. In seeking an implementation to modify, we considered SABUL, UDT, Tsunami, RBUDP, Net100[22] and RudeTCP. Our lack of expertise in kernel-level coding ruled out Net100 and RudeTCP, so we downloaded and tested the software for the four application-level implementations. Among these we found the SABUL code the easiest to modify and hence selected it as the basis for FRTP.

III. A MODIFIED VERSION OF SABUL FOR DEDICATED END-TO-END CIRCUITS: FRTP

In this section we first briefly review the SABUL protocol. Then we consider the design of FRTP and the modifications needed in SABUL to achieve these design goals.

A. SABUL Overview

SABUL is a rate-based protocol designed for use on the Internet where the sender senses the available bandwidth and adjusts its sending rate accordingly. SABUL is implemented as an application-level process using a combination of UDP and TCP. Each SABUL session has two channels: a UDP data channel and a TCP control channel. The UDP data channel is used for the actual user-data transfer from the sender to the receiver.

The TCP control channel is used for control-information (control messages for congestion control, flow control, and error control) transfer from the receiver to the sender.

SABUL uses a rate-based scheme to tune the packet sending rate, addressing both the flow control (receive-buffer overflow) and congestion control (network switch buffer overflow) problems. The sending rate is controlled in the SABUL implementation by adjusting the inter-packet transmission time. The sending rate is updated periodically by the sender according to the packet-loss rate. There is also a maximum flow window at the sender which limits the number of unacknowledged packets in the network in case control packets on the TCP path get delayed or lost.

SABUL uses a selective-ARQ error control scheme, in which only the lost packets are retransmitted, to achieve high efficiency. Negative Acknowledgements (NAKs) from the receiver are used to report packet loss, while positive Acknowledgements (ACKs) are used to update the sender's retransmission buffer and retransmission timer. The retransmission timer at the sender is not maintained per-packet. Instead, it is a mechanism to detect and recover from conditions that lead to long periods of silence from the receiver.

B. FRTP Design

SABUL provides a good basis to develop a new transport protocol for dedicated end-to-end circuits. However, since SABUL was originally designed for packet-switched networks, it includes many features that are not only unnecessary for dedicated end-to-end circuits, but have an adverse effect on the performance of data transfer on circuits. In this sub-section, we will discuss the differences between our design rationale and SABUL's, and describe the corresponding modifications we made in SABUL.

1) Congestion Control

Congestion control is defined as any scheme by which the sender limits the sending rate to prevent too much traffic in the network. As described in Section II, congestion control is not required on end-to-end circuits during the data transfer. Congestion is handled during the circuit setup.

SABUL defines a SYN control packet, which is sent from the receiver to the sender periodically. This packet contains the number of received packets since the last SYN event and serves to detect congestion. Upon receiving a SYN control packet, the sender performs congestion recovery by using the recent packet-loss rate to update the sending rate.

We removed the SYN generation function from the SABUL receiver code and the sending rate update function from the SABUL sender code. The receiver notifies the sender with a user-specified sending rate prior to the data transfer. The actual sending rate then stays unchanged during the whole data transfer.

2) Flow control

There are three well-known flow-control methods: ON/OFF, window-based, and rate-based. In ON/OFF and window-based flow control schemes, the receiver sends messages to control the behavior of the sender dynamically. These receiver-based flow control schemes are inefficient on end-to-end circuits because the circuit lies idle while the sender waits for an ON or "window open" signal from the receiver. This leads to poor circuit utilization. In circuit-switched networks, since resources are reserved in the network, we would like to send the data at the fixed circuit rate, thus fully utilizing it. This can be considered as rate-based flow control with the rate held fixed. At first glance it appears that with a dedicated circuit one could simply match the sending rate to the receiver rate (which is ideally equal to the circuit rate), thus eliminating the need for receiver buffers. For example, on end-to-end telephone circuits, senders (speakers) generate audio data at the same rate at which receivers (listeners) consume data. However, unlike telephones that perform their single dedicated function, general-purpose computers that are typically involved in file transfers engage in several other activities concurrent with file transfers. This requires the operating system to schedule various tasks in and out of the processor as needed, which implies that data received on a NIC

is not moved at a guaranteed constant rate from the NIC to the disk. Furthermore disk access rates are not constant. There can be significant variability based on the location to which data needs to be written. Variability also arises at the sender. Based on when the operating system at the sender schedules the network-related kernel threads and drivers, data is moved from disk to memory (user-space and/or kernel-space) and from there to the NIC at varying rates. Thus we have 2 conflicting requirements. On the one hand we would like to fully utilize the reserved circuit bandwidth by sending data at the fixed circuit rate, potentially causing receive buffer overflow, packet loss and retransmission. On the other hand we would like to minimize packet loss at the receiver by adjusting the sending rate to match the varying receiver rate, potentially leaving the circuit underutilized.

In SABUL, the flow control scheme is clubbed with the congestion control scheme. Receive buffer overflow events are also indicated by the SYN packets described earlier. In addition to the sending rate, which is maintained by adjusting the inter-packet transmission time, the SABUL protocol describes a flow-control window which increases in a slow-start fashion till it reaches a maximum size. The implementation, though, does not have the slow-start scheme, using instead, a fixed window size.

We decided to fix the sending rate to the circuit rate so that the circuit utilization is high. The circuit rate (hence, the sending rate) should not be chosen to be so high that excessive receive buffer overflow and retransmissions increase the transfer time and reduce goodput. Neither should a pessimistically low circuit rate be selected- even though it keeps the circuit utilized and minimizes receive buffer overflow- since the goodput again suffers because of the excessive delay of data transfer. As we show through experiments later in this paper, the circuit rate should be chosen carefully to achieve acceptably high circuit utilization and goodput and low receive buffer overflow.

3) Error control

Error control algorithms involve error detection, error reporting and error correction. Two main schemes for error control can be identified- sender error detection through time-outs and receiver error detection through out-of-sequence packet reception. On end-to-end circuits either of these schemes can be used, unlike in packet-switched networks, because of the guaranteed in-sequence delivery on circuits. The overhead of maintaining timers at the sender, especially in an application level implementation like ours, made us choose the receiver error detection scheme. Errors are reported using NAKs. However, ACKs are still used because the sender needs to update its retransmission buffers. We will need retransmission buffers, where data is held until acknowledged, at the sender to avoid repeated disk accesses. Retransmission buffers cannot be too large because of the limited memory size at end hosts. Therefore we need ACKs to confirm the delivery of packets and allow the release of the corresponding space in the retransmission buffer. For error correction we propose to use the selective Automatic Repeat reQuest (ARQ) scheme.

SABUL, even though it was designed for packet-switched networks, uses receiver error-detection. ERR packets are sent by the receiver, whenever packet loss is detected or periodically, with the list of sequence numbers of lost packets. The receiver also sends ACK packets periodically acknowledging all packets till the sequence number sent in the ACK packet.

The error control scheme used by SABUL matches our design goals for FRTP, so we retained it.

4) Use of dual communication paths

In our CHEETAH solution, to achieve a high utilization of the circuit-switched network, we propose using a unidirectional circuit from the server to the client (since this is the primary direction of data flow). However, this raises the question of how to transport reverse-path control messages, such as ACKs, NAKs and any control messages. For example, our protocol design requires a control message exchange to send the sending rate to the sender prior to the actual data transfer. If a dedicated end-to-end circuit is used for this exchange, utilization will suffer since control packets are expected to be few in number.

Hence we propose to equip an additional NIC into end hosts and use a TCP connection, set up via the Internet through this secondary NIC, for such exchanges. In other words, our transport solution is a combination of a dedicated end-to-end circuit for data transfer and a TCP/IP path for control message transfer.

SABUL uses a UDP data channel from the sender to the receiver and a TCP control channel in the opposite direction. But it has no provision to use separate NICs for the 2 separate channels.

In FRTP, we separated the two channels making them go through separate NICs (network interfaces). User data is transferred on the unidirectional circuit via the primary NIC, while the reverse-direction control messages are transferred over the Internet, using TCP, via the secondary NIC.

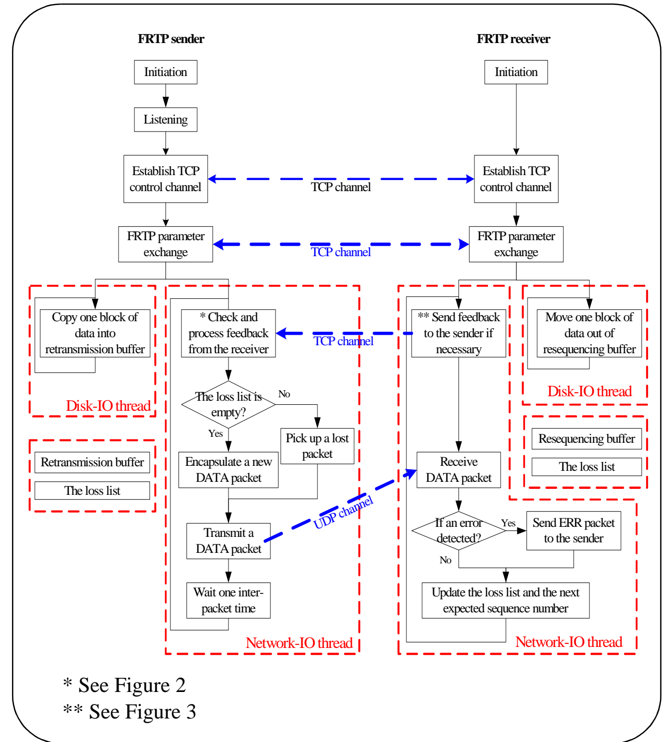


Figure 1. Data Sending/receiving Procedure in FRTP

IV. IMPLEMENTATION OF THE NEW TRANSPORT PROTOCOL

Based on the discussion in Section III, we modified the SABUL implementation to support our design requirements and removed all unnecessary functions. As shown in Figure 1, a FRTP session starts with a TCP connection establishment between the sender and the receiver. The FRTP sender opens a TCP listening port and waits for an incoming connection attempt. A TCP connection is established upon receipt of a request from the FRTP receiver. The sender and receiver then exchange a set of FRTP parameters via the TCP connection, such as the user-specified sending rate and UDP data channel's port number.

After a successful TCP control channel establishment and FRTP parameter exchange, the data-transfer on the end-to-end circuit starts via the UDP socket[2]. During the whole data transfer, the sender is responsible for data transmission and retransmission based on feedback from the receiver. A select/poll mechanism is used by the sender to handle two threads, a network-I/O thread and a disk-I/O thread, simultaneously. The disk-I/O thread copies data from the disk (in disk-to-disk transfers) or upper-layer application buffer (in memory-to-memory transfers) into the FRTP sending buffer. This buffer is also used as a retransmission buffer for error control purposes. The disk-I/O thread copies the user data block by block into the retransmission buffer till it is full. In case the buffer is full, the disk-I/O thread waits for a fixed time interval before trying again. Meanwhile, the network-I/O thread reads data from the retransmission buffer and encapsulates it into FRTP DATA packets. It sends one DATA packet every inter-packet transmission time via the UDP socket. The sender maintains a list to record the sequence numbers of lost DATA packets. Every inter-packet transmission time, the sender checks the loss list first. If it is not empty, the DATA packet with the smallest sequence number in the loss list is retrieved from the retransmission buffer and retransmitted. Otherwise, the network-I/O thread encapsulates a new DATA packet and sends it out on the network. The new transmitted data will be kept in the FRTP retransmission buffer until the corresponding acknowledgement is received. Every inter-packet transmission time, the network-I/O thread also checks for ACK and ERR packets from the receiver, as illus-

2.UDP is a connectionless protocol, so there is no connection establishment procedure for UDP data channel.

trated in Figure 2. The DATA packets that are acknowledged by ACK packets are released from the FRTP retransmission buffer, while the sequence numbers of lost packets reported by ERR packets are inserted into the loss list. To limit the number of unacknowledged packets in the network, in case the control packets are lost over the TCP channel, the sender maintains an EXP timer. For simplicity, instead of maintaining timers for each DATA packet, the FRTP sender maintains one common timer for all recently transmitted DATA packets. If an ACK or ERR is not received before the EXP timer expires (1s time-out value in our implementation), all outstanding DATA packets are inserted into the loss list and retransmitted.

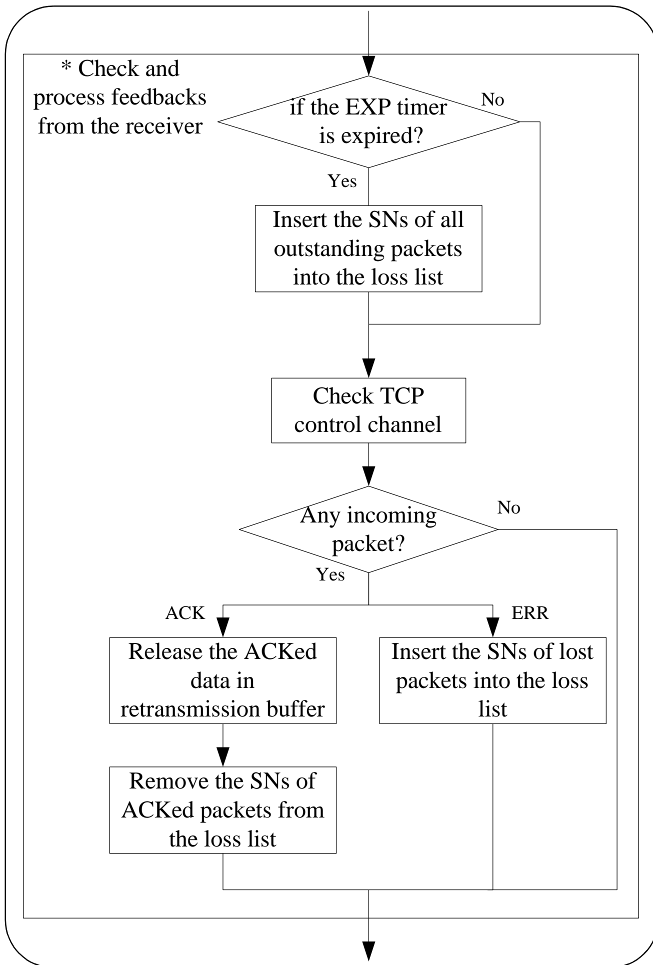


Figure 2. Feedback Checking and Processing at the FRTP Sender

Similarly, at the receiver side, a network-I/O thread receives and processes DATA packets. The data payloads are written into the FRTP resequencing buffer to be accumulated together into blocks. The data blocks in the resequencing buffer are then copied to the disk (in disk-to-disk transfers) or the upper-layer application buffer (in memory-to-memory transfers) by a disk-I/O thread. The receiver also maintains a packet loss list. The sequence number of each lost packet is kept in the loss list until a correctly retransmitted copy is received. Packet loss is detected by comparing the received sequence number to the next expected sequence number. If packet loss is detected, the sequence numbers of the lost DATA packets are inserted into the loss list, and an ERR packet containing the sequence numbers of the lost DATA packets is sent back to the sender immediately. The retransmitted DATA packets could also be lost due to link errors and receive-buffer overflows. To ensure the reliable delivery of DATA packets, the whole loss list, if it is not

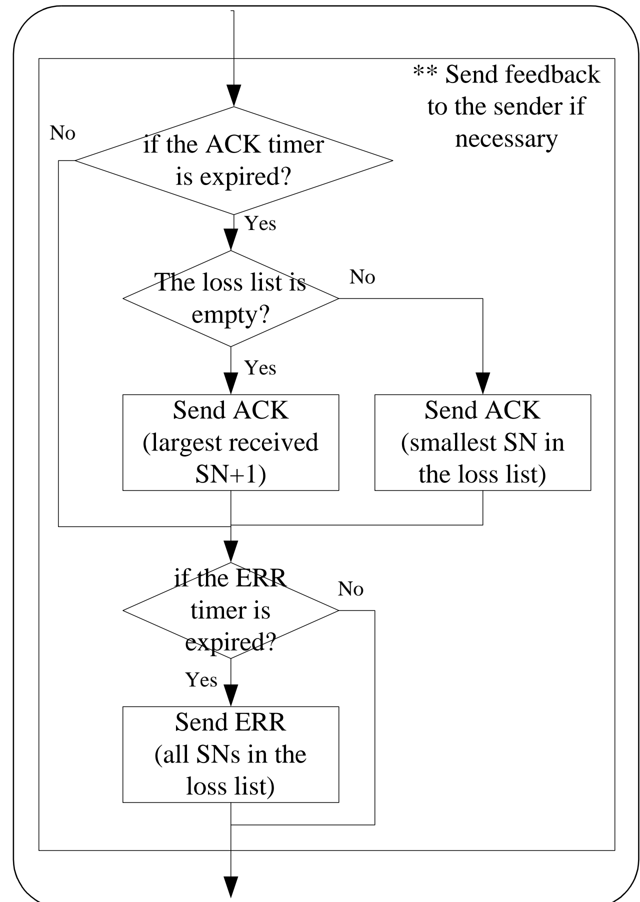


Figure 3. Feedback Sending at the FRTP Receiver

empty, is sent back periodically (every 20ms in our implementation) as an ERR packet. ACK packets are sent back to the

sender periodically (every 100ms in our implementation), as illustrated in Figure 3, or immediately when FRTP resequencing buffer is full[3]. The sequence number carried in ACK packets is the next expected sequence number, which equals the largest received sequence number plus 1 or the minimum sequence number in the loss list, if the loss list is not empty.

V. EXPERIMENTAL RESULTS

One of the key features of FRTP is to have the sender transmit data at a constant rate that corresponds to the circuit rate. The constant sending rate is realized by controlling the inter-packet transmission time. We conduct an experiment to test the effectiveness of rate-based flow control in the FRTP implementation.

Another purpose of our experiment is to study how to select an appropriate sending rate (circuit rate). We realize that a higher sending rate does not always produce a higher throughput due to the limitation of the end hosts. To select an appropriate rate, we must take into account a number of factors, such as the scheduling scheme of the operating system, the hard disk access rate, UDP buffer size, MTU size, and other FRTP related parameters. Although some of these factors, such as the scheduling scheme of the operating system and the hard disk access rate, cannot be controlled, other parameters can be adjusted to achieve better performance. We test the impact of some of these parameters in the experiments.

In our experiments, we connected two Dell Precision 650 workstations via a Dell PowerConnect Gigabit Ethernet switch. Each Dell workstation has a 2.4-GHz Intel Xeon™ CPU connected to a 533-MHz front-side bus (34Gbps CPU bandwidth), an E7505 chipset with 512MB of DDR 266MHz memory (17Gbps memory bandwidth), an 80GB ATA/100 7200 RPM EIDE disk drive with 2MB cache (400Mbps average writing rate measured by Bonnie [23]), and two 64bit/100MHz PCIx GbE NIC (6.4Gbps network bandwidth). The operating system on both workstations is RedHat Linux 9 with version 2.4.20-30.9 kernel. Tcpdump is used to better facilitate the analysis of data transfers [24].

Our experiments focus on the performance of bulk data transfers. We ran FRTP applications on both workstations and transferred a 127MB file between them. No other user processes were running on either workstation during the experiment.

A. Results with Default Settings

We began the experiments with default FRTP settings: 256KB UDP buffer size, 1500Bytes MTU size, 40MB FRTP buffer size, and 8MB block size for disk I/O operations. The sending rate is increased from 50Mbps to 1Gbps. Figure 4 shows packet-loss rate and transfer throughput (note that we use the term "throughput" to denote "goodput" in the whole of section V) as a function of sending rate, for both disk-to-disk and memory-to-memory transfers.

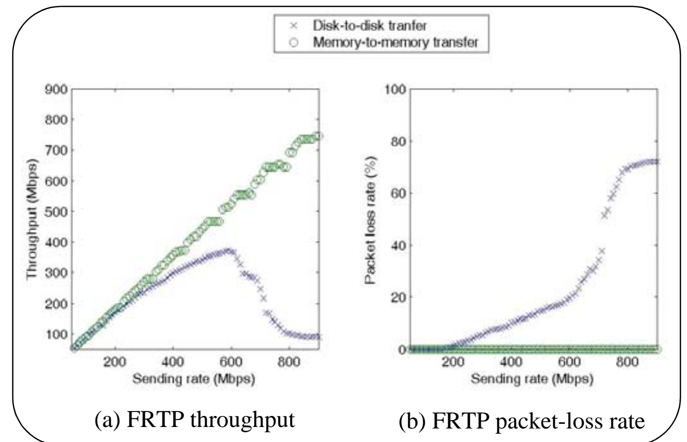


Figure 4. Packet-Loss Rates and Throughputs vs. the Sending Rate in FRTP Experiments (DATA Packet Size=1500B, UDP Buffer Size=256KB, FRTP Buffer Size=40MB, FRTP Data Block Size=8MB)

For disk-to-disk transfers, as the sending rate is increased, the plots show the throughput increasing as expected with zero or a small packet-loss rate. When the sending rate reaches a certain level (~200Mbps in Figure 4), the packet-loss rate becomes significant. This happens because of the limitations of the end-host hardware and software modules that are involved in moving blocks of the file from disk into the FRTP buffer, UDP buffer,

3.This could happen when the sender sends the data in a burst at a rate higher than the specified sending rate.

and finally into the NIC at the sender, and similarly through the NIC, UDP buffer, FRTP buffer, and disk at the receiver. The major bottleneck in our experimental configuration is the hard disk access rate, especially the writing rate. Even though the measured average writing rate of our disks is around 400Mbps, its worse-case writing rate might be much lower than the average value. For example, the disk driver requires a certain amount of time to switch between cylinders and disk heads. During this switching time, the disk read/write operations have to be suspended. To better reflect a disk's real-time performance, the concept of "disk sustained transfer rate" was introduced. This rate is dependent on a disk's media transfer rate, but includes the overhead of cylinder and head switching. Based on manufacturer's specification of our disk, and other similar products, a rough approximation of the sustained transfer rate for our disk is 200Mbps. This explains the significant packet loss rate when the sending rate is larger than 200Mbps.

Increasing the sending rate beyond 200Mbps leads to excessive packet loss because of receive buffer overflows, causing retransmissions that impact overall throughput. As a result, the throughput slowly reaches an "optimal" value (~370Mbps in Figure 4) at a 590Mbps sending rate and then decreases. This "optimal" value approximates the 400 Mbps average disk access rate, the expected bottleneck in the experiment. The small difference is caused by the additional processing overhead incurred in handling packet loss. The results for the memory-to-memory transfer experiments are also shown in Figure 4. By removing the effects of disk access, which is the bottleneck in the disk-to-disk transfer, we can achieve a higher throughput (up to 910Mbps) without incurring significant packet losses.

To check the effectiveness of the FRTP implementation in maintaining a constant sending rate (equal to the circuit rate), we captured several tcpdump trace files while running FRTP file transfers. The actual inter-packet transmission times seen on the link were retrieved from the trace files. Figure 5 shows an example of inter-packet transmission times in a FRTP file transfer at a 50Mbps sending rate. The plot shows that the variance of actual inter-packet transmission times is very small. The standard variance of inter-packet transmission times is only 0.00005, which is quite acceptable considering the inevitable variability at the sender. Due to the limitations of the machine running Tcpdump, we could not collect measurement data at very high sending rates, but we expect similar behavior.

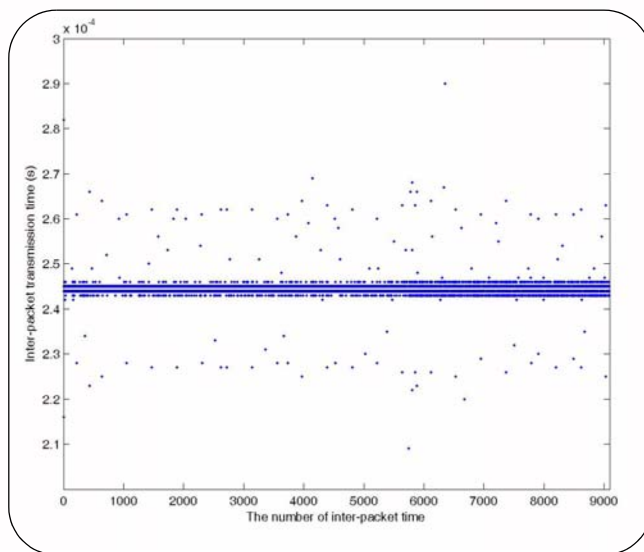


Figure 5. An Example of Inter-packet Transmission Times within a FRTP File Transfer (Sending Rate=50Mbps, DATA Packet Size=1500B, UDP Buffer Size=256KB, FRTP Buffer Size=40MB, FRTP Data Block Size=8MB)

Another important consideration is the CPU usage of FRTP. As an application-level implementation, FRTP consumes a large number of CPU cycles; so its performance is easily compromised by other concurrent processes in a multitasking environment. CPU utilization of FRTP was measured as follows: Linux has system calls, "times()", which reports the number of CPU clock ticks used by the calling process and all its children and "gettimeofday()", which reports the current time. We call "times()" in the FRTP code to obtain the number of CPU clock ticks used by FRTP and then divide the value by the CPU frequency to get FRTP's CPU usage in seconds. We also use "gettimeofday" to measure the total time for which FRTP runs. CPU utilization of FRTP is then calculated as CPU time used by FRTP over real time for which FRTP runs.

Figure 6 plots CPU utilization versus sending rate in FRTP file transfers. The plot shows that CPU utilization at the sender

is always greater than 70%, and it decreases as sending rate is increased. The high CPU utilization is explained by the fact that, after transmitting a packet, the sender process bides the remaining inter-packet transmission time interval by busy-waiting. The inter-packet transmission times are on the order of microseconds (1500B*8/500Mbps = ~24microseconds for 500Mbps sending rate) and busy-waiting seems to be the only option we have with a general OS. On the receiver side, CPU utilization is relatively lower and increases with sending rate. Such high CPU hogging is a major drawback of application-level implementations, since it leaves little time for the application to do any computation (though this is a non-issue for bulk-data transfer), and adversely affects/gets affected by other concurrent CPU-intensive processes. For instance, when we start a Matlab process at the sender while running FRTP with a 500Mbps sending rate, the FRTP throughput immediately dropped from 340Mbps to about 80Mbps.

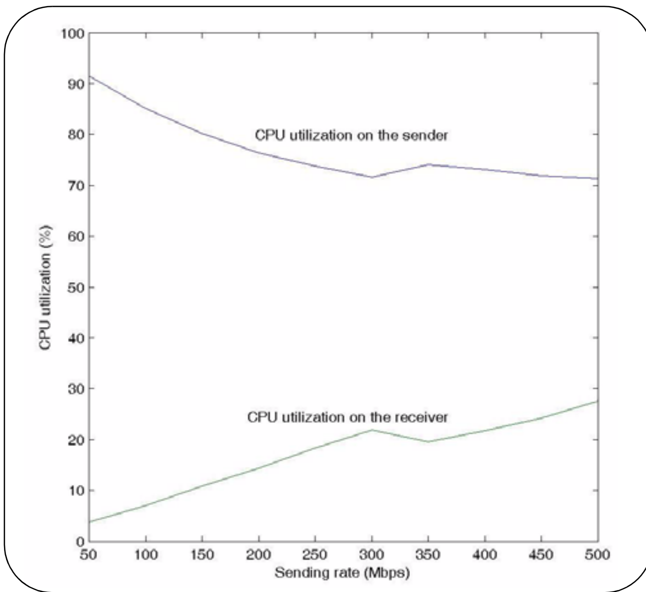
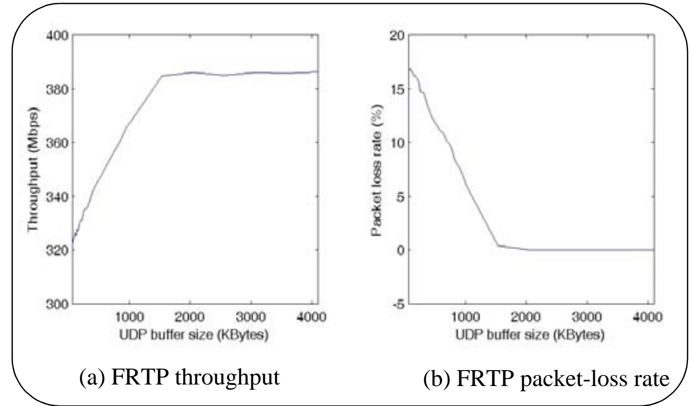


Figure 6. CPU Utilization vs. the Sending Rate in FRTP Experiments (DATA Packet Size=1500B, UDP Buffer Size=256KB, FRTP Buffer Size=40MB, FRTP Data Block Size=8MB)

B. Impact of UDP Buffer Size

It is well known that TCP throughput is improved by properly selecting TCP buffer size. In this experiment we test whether a similar improvement in FRTP performance can be achieved by increasing the UDP buffer size.

For this experiment we fixed the sending rate at 500Mbps and measured FRTP performance with different UDP buffer sizes. UDP buffer size was changed from 64KB to 4MB by calling the system function setsockopt(). Default values were used for all other FRTP parameters. We plot packet loss and transfer throughput versus UDP buffer size in Figure 7.



(a) FRTP throughput (b) FRTP packet-loss rate

Figure 7. Packet-Loss Rates and Throughputs vs. UDP Buffer Size in FRTP Experiments (DATA Packet Size=1500B, Sending Rate=500Mbps, FRTP Buffer Size=40MB, FRTP Data Block Size=8MB)

As the UDP buffer size is increased, Figure 7 shows that FRTP throughput increases while the packet-loss rate decreases. For example, with a 2MB UDP buffer, the average throughput at a 500Mbps sending rate increases to 386Mbps (a 20.6% improvement from 320Mbps with the 64KB UDP buffer) and the loss rate drops from 16.96% to 0.03%. At the FRTP sender, a small UDP sending buffer increases the number of memory copies from the FRTP buffer to the UDP buffer. This causes a serious degradation of FRTP performance in an environment where the CPU resource and/or the bus speed is the bottleneck. At the receiver, a small UDP receive buffer incurs unnecessary CPU overhead and data movement delays. Furthermore, a small UDP receiving buffer increases the possibility of receive-buffer overflows due to variations in the data movement rate from the UDP buffer to the FRTP buffer. When the operating system is temporarily unable to schedule system resources to move the received data out of the UDP buffer, a small UDP receiving buffer can overflow, especially in high-speed data transfers.

By removing the UDP buffer size limitation, the throughput gradually approaches the expected maximum value which is

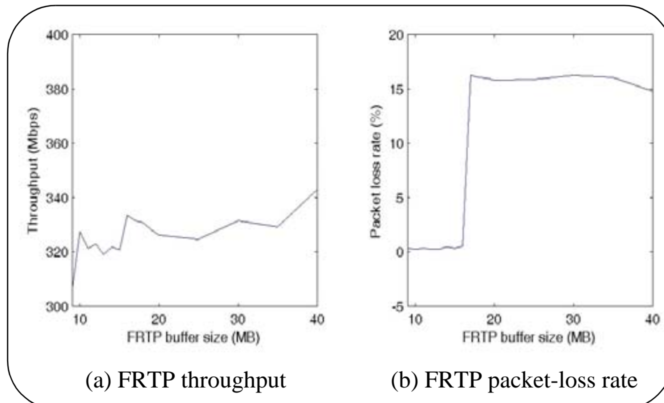
upper bounded by the average disk writing rate. The highest throughput value seen in the experiment is 400.03Mbps, which matches very well with the 400Mbps average disk writing rate measured by Bonnie. Compare this plot with Figure 4. (obtained using default settings), in which we could only achieve a maximum throughput of 370Mbps, at a 590Mbps sending rate, with a 19% packet-loss rate. We conclude that a larger UDP buffer size does improve FRTP performance.

However, increasing the UDP buffer size does not always bring us benefits. There is no obvious improvement seen when we further increase the UDP buffer size beyond 2MB. This is understandable because with a large UDP buffer, the only bottleneck is the disk writing rate, and thus, increasing the UDP buffer size will not help improve throughput. UDP buffer size should be tailored for each transfer. In our particular case, a UDP buffer size slightly higher than 2MB produced "optimal" results.

C. *Impact of FRTP Buffer Size*

In this set of experiments, the sending rate was fixed at 500Mbps. All parameters were set to default values except the FRTP buffer size. We changed the FRTP buffer size from 9MB to 40MB and measured FRTP performance. Figure 8 plots packet-loss rate and transfer throughput versus FRTP buffer size.

As FRTP buffer size increases from 9MB to 40MB, FRTP throughput increases from 305Mbps to 342Mbps, a 12.1% improvement. In our particular experiment, the "optimal" FRTP buffer size is around 16MB although a slightly higher throughput value can be seen with larger FRTP buffer sizes. Increasing the FRTP buffer size beyond 16MB brings little benefit because of the dramatic increase in packet-loss rate. This is reasonable because a very large FRTP buffer will consume a large amount of system resources for memory management. Again, the "optimal" value of FRTP buffer size depends on the particular hardware and software configurations, and should be tailored for each transfer.



(a) FRTP throughput (b) FRTP packet-loss rate

Figure 8. Packet-Loss Rates and Throughputs vs. FRTP buffer size in FRTP Experiments (DATA Packet Size=1500B, UDP Buffer Size=256KB, Sending Rate=500Mbps, FRTP Data Block Size=8MB)

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented the design and implementation of Fixed Rate Transport Protocol (FRTP), a transport protocol for dedicated end-to-end circuits. FRTP is a modified version of SABUL. Two features distinguish it from other transport solutions: 1) data is transmitted at a fixed rate (matched to the circuit rate) to keep the circuit fully utilized; 2) it uses dual communication paths, a unidirectional end-to-end circuit for data transfer and the Internet for reverse control message transfer. We implemented FRTP as an application-level process and carried out a series of experiments, in our local testbed, to quantify its performance. One future action item for our work is to duplicate the experiments in a "real", wide-area network.

The experimental results show that the FRTP implementation is effective in producing a constant sending rate during the data transfer. The inter-packet transmission times seen on the wire support this claim. Thus, the most important objective of our transport protocol work- a fixed sending rate- is successfully achieved.

The experimental results also show that FRTP is able to achieve high throughput. In our disk-to-disk experiments, FRTP successfully achieved the theoretical maximum throughput, a 400Mbps disk access rate in our experimental configuration. We plan to repeat the experiment using high-performance

RAID disks. In memory-to-memory transfer experiments, by removing the disk access bottleneck, we saw a throughput of up to 910Mbps without much packet loss.

We also notice from the experiments that several configurable parameters, such as UDP buffer size and FRTP buffer size, have a significant impact on performance. To obtain optimal results, these parameters have to be carefully tuned. In most cases FRTP performance improves significantly by using non-default values for the parameters. We now plan to consider the affect of other parameters, like packet size and disk read/write block size, on FRTP performance, and also fully understand these affects.

FRTP throughput increases, till a limit, as sending rate is increased. However, due to the variability of the receiving capability, receive buffer overflow and the corresponding packet loss cannot be completed avoided. Therefore FRTP throughput is always lower than the sending rate. Moreover, bandwidth utilization- the ratio of goodput and circuit rate- does not increase linearly with the sending rate. So it is necessary to strike a balance between throughput and utilization of the reserved resources.

One disadvantage of FRTP is its high CPU utilization. One should avoid using FRTP concurrently with other CPU-intensive processes. If the variability of the receiving capability is severe due to resource contention between concurrent processes, the throughput and circuit utilization would drop to an unacceptable level. This is the biggest problem of running application-level implementations on end hosts running general-purpose operating systems. We are trying to improve the FRTP implementation so as to make it less CPU-hungry. On the other hand, transport protocols designed for OS-bypass implementations, such as ST [25], use much lesser of the CPU resources, which makes them more attractive in environments where other CPU-intensive processes are running. We will explore those OS-bypass transport protocols in future work.

VII. REFERENCES

[1] M. D. Brown, "Blueprint for the future of high-performance networking introduction," *Communications of ACM*, Vol. 46, No. 11, pp. 30-33, Nov. 2003.

[2] T. DeFanti, C. D. Laat, J. Mambretti, K. Neggers, and B. St. Arnaud, "TransLight: a global-scale LamdaGrid for e-science," *Communications of ACM*, Vol. 46, No. 11, pp. 34-41, Nov. 2003.

[3] I. Foster et al, "Grid services for distributed systems integration," *IEEE Computer*, June 2002, pp. 37-46.

[4] "Abilene," http://abilene.internet2.edu/.

[5] "The Energy Sciences Network, (ESnet)," http://www.es.net/.

[6] "Teragrid," http://www.teragrid.org/.

[7] D. Borman, R. Braden, V.Jacobson, "TCP Extensions for High Performance," *RFC 1323,* May 1992.

[8] S. Floyd, "HighSpeed TCP for Large Congestion Windows," *IETF RFC 3649*, December 2003.

[9] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," *Proc. of IEEE Infocom 2004*, March 2004.

[10] T. Kelly, "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks," *PFLDnet 2003*, http://datatag.web.cern.ch/datatag/pfldnet2003/, February 3-4, 2003, Geneva, Switzerland.

[11] P. Varaiya and J. Walrand, "High-Performance communication Networks," *Morgan Kaufmann*, San Francisco.

[12] Y. Gu, X. Hong, M. Mazzucco, and R. L. Grossman, "SABUL: A High Performance Data Transfer Protocol," submitted to *IEEE COMMUNICATIONS LETTERS*.

[13] Y. Gu and R. L. Grossman, "End-to-End Congestion Control for High Performance Data Transfer," submitted to *IEEE/ACM Transaction on Networking*.

[14] Tsunami, http://www.indiana.edu/~anml/anmlresearch.html.

[15] E. He, J. Alimohideen, J. Eliason, N. K. Krishnaprasad, J. Leigh, O. Yu, and T. A. DeFanti, "Quanta: A Toolkit for High-Performance Data Delivery over Photonic Networks," *Future Generation Computer Systems*, 1005, 2003.

[16] "CANARIE's CA*net 4," http://www.canarie.ca/canet4/.

[17] "OMNInet," http://www.icair.org/omninet/.

[18] "SURFnet," http://www.surfnet.nl/en/.

[19] "UKlight," http://www.ja.net/development/UKLight/.

[20] M. Veeraraghavan, X. Zheng, H. Lee, M. Gardner, and W. Feng, "CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture," *Proc. of Opticomm 2003*, Dallas, TX, Oct. 13-17, 2003.

[21] E. He, J. Leigh, O. Yu, and T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *Proc. of the IEEE Cluster Computing 2002*, pp. 317-324, Chicago, Illinois, September 23-26, 2002.

[22] http://www.csm.ornl.gov/~dunigan/net100/.

[23] "Bonnie," http://www.textuality.com/bonnie/.

[24] "TCPDUMP Public Repository," http://www.tcpdump.org.

[25] ANSI, "Information Technology - Scheduled Transfer Protocol (ST)," T11.1/Proj. 1245-M/Rev 4.0, October 2000.