Simulation and Analysis of PDU Traffic Bundled under Packet Alloying

Juan J Vargas-Morales
Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
San José, Costa Rica
jvargas@ecci.ucr.ac.cr

ABSTRACT

Packet Alloying, a technique for the optimized bundling of packets, is proposed. It produces aggregations that preserve the internal Protocol Data Unit (PDU) format, allowing the resulting packets to be subjected to further bundling or compression by conventional techniques. Using logs from vignettes simulated by the OneSAF Testbed Baseline (OTB), a simulator is used to analyze PDU traffic over a wireless flying Local Area Network aboard airplanes, including a satellite and a ground station. Applying Packet Alloying during the simulation of an example vignette, a reduction in the magnitude of negative slack time from -75 to -9 seconds for the worst spike was achieved.

Contributions of this research include the formalization of a selective PDU bundling scheme, and the study of different predictive algorithms for the next PDU. These results demonstrate the validity of packet optimizations for distributed simulation environments and other possible applications such as TCP/IP transmissions.

Categories and Subject Descriptors

I.6.6 [**Simulation and Modeling**]: Simulation Output Analysis; E.4 [**Coding and Information Theory**]: Data compaction and compression

General Terms

Experimentation, Measurement, Performance

Keywords

DIS protocol, PDU, Packet Alloying, bandwidth simulation, network traffic

1. INTRODUCTION

The Distributed Interactive Simulation (DIS) protocol is defined in the IEEE family 1278 of Standards [4]. DIS is primarily used for military simulations of synthetic battlefields including, weapons, vehicles, terrain information, friend and enemy forces, etc. such that the final simulation is as realistic as possible. It is a stateless system not based on servers that utilizes reliable multicast communications with autonomous nodes using dead reckoning to calculate the position of all entities in the simulation. Using a workstation connected to the network, training personnel interact with each other, playing different roles, either as part of the friend or the enemy forces. DIS is relatively old. It was developed in 1993 during a series of workshops held at the Institute for Simulation and Training of the University of Central Florida. Its successor, the High Level Architecture (HLA) protocol, took its place for most military simulations. However, DIS continues being researched and improved in other areas like USAF's Distributed Mission Operations Center (DMOC), video games, space exploration and medicine.

The fundamental communication packets under DIS are the Protocol Data Units (PDUs). PDUs carry all the information relevant to the simulation, like data about position, speed, direction of movement, type of vehicle, type of weapons, damages, status, etc. Due to the intended realism, the DIS protocol requires high network bandwidths, making it too heavy for less serious simulations like multi-player online video games. There is a simpler version of DIS called DIS-Lite [8] which is more appropriate for such applications. DIS-Lite offers several advanced features, including packet bundling, latency compensation and enhanced dead reckoning algorithms tailored for air vehicles.

DIS defines 27 types of PDUs. However, the standard is open for new types. PDUs contain different internal fields, starting with a header that specifies the exercise identification number, protocol version, type of PDU, protocol family, timestamp and length. Fields following the PDU header are specific to each type and of a variable size. In the studied vignette PDU lengths ranged form 26 to 1368 bytes. It was observed that most transmitted PDUs contained zeros in many fields, causing a waste of bandwidth due to transmissions of long sequences of zero bits that can be alleviated by PDU compression and/or bundling.

In 1998, the U. S. Army Simulation, Training and Instrumentation Command (STRICOM) started to develop a recommendation of the Semi-Automated Forces (SAF) system to be used as the baseline for the integration of a One Semi-Automated Forces (OneSAF) Testbed Baseline (OTB), which is a computer simulator of the battleground based on DIS. Detailed information about the historic development of OTB is found in [1, 6]. OTB requires high bandwidth connections, as in a Local Area Network (LAN), to

LANC '07, 10-11 October 2007, San José, Costa Rica

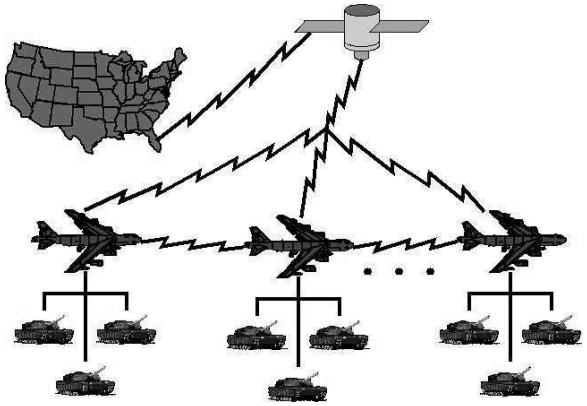

Figure 1: The flying network.

produce a realistic simulation. However, the usage of OTB simulations onboard airplanes for mission planning and rehearsal while enroute to deployment was considered. The initiative is called Joint En-route Mission Planning and Rehearsal System for Near-Term (JEMPRS-NT). The scenario includes some airplanes flying in formation carrying a wired LAN of several workstations onboard and wireless communications among planes. Also, a CONtinental US (CONUS) ground station participates in the exercise through a satellite link, involving three types of connections: wired, wireless and satellite. Due to its nature, each connection has a different bandwidth capacity, showing the satellite link the slowest one. Figure 1 depicts the scenario.

The actual scenario had not been built and tested in 2004, at the time of this research. Consequently, there was a consideration about the performance of OTB simulations through the wireless and satellite connections. A network simulation of the traffic was conducted to determine the bandwidth requirements of military mission rehearsal activities while enroute to deployment [13, 12, 11, 10]. In preparation for the simulation, a vignette was defined and run under OTB, capturing logged packets to be used as input data. The information logged included the type, length, and timestamp of each PDU. The network simulator was based on the OMNeT software [9] to asses the bandwidth in each link. Each simulated workstation includes a PDU generator module and a sink module that consumes arrived PDUs. Instead of using a PDU random generator of a predefined probabilistic distribution, PDUs captured from OTB were arranged in the generator modules of the simulator and broadcasted to sink destinations at times specified by OTB timestamps. Hence, the network simulator was reproducing the OTB traffic behavior as accurate as possible.

By analyzing the traffic behavior, it was possible to observe bottlenecks in several segments of the network under different bandwidths. At low speeds of the wireless links, bottlenecks started to build up in satellite and router queues. Because the speed of satellite links is considered static, a special type of bundling and aggregation of PDUs called Packet Alloying [10] was proposed and evaluated with success.

2. OFFLINE AND ONLINE ALGORITHMS

General offline and online algorithms for predictive environments have been studied in the literature for a considerable time [5, 2, 3]. Related to network traffic, all the packets sent from an origin to a destination form a sequence. If two or more consecutive PDUs contain the same data in some fields, they could be bundled into one single packet, saving header and data bytes during transmission.

Due to the real-time nature of the simulation, once a given packet is ready for transmission, the decision of waiting and bundle the next one to it, or send it without delay has to be taken online, without knowledge of the next PDU content. Thus, bundling strategies are confronted with the decision of waiting for the next packet to arrive and bundle it to the current block, or sending the current block and start a new collection of packets from scratch. *Online algorithms* are characterized by this lack of knowledge about the future. They decide the next action based only on the already processed PDUs. Phillips [7] indicates that an online algorithm receives each input in sequence and must process it immediately, serving the sequence of requests one item at a time without having explicit knowledge of the following inputs.

On the other hand, we could capture all the PDUs and, once the OTB simulation is over, analyze the sequences of packets, deciding if a PDU can be bundled to other consecutive packets based on a complete knowledge of the sequence. If the selected action is taken using the complete knowledge of the whole sequence, the selection process constitutes an *offline algorithm*. Offline algorithms have access to all the past and future sequences of PDUs in advance. Therefore, it is possible to find the best offline algorithm that makes only optimal decisions. A decision is optimal if it minimizes some *cost function*. In the transmission of network packets, the cost function could be the total latency time incurred by all the packets sent from the origin to the final destination, or the absolute value of the sum of all negative slack times, as defined below.

If the generator module of the simulator reads the next PDU at simulation time *Tread* and is timestamped at time *Tstamp*, the difference *Tstamp - Tread* is called *slack time*. If it is positive, the network traffic is light at that moment, the simulator is ahead of time and can rest until time *Tstamp* to send the PDU. But if the difference is negative, the simulator is in trouble, it is behind the schedule due to a heavy loaded traffic and needs to send the PDU as soon as possible.

The main application of offline algorithms in this research lies in the possibility of comparing them to the corresponding online counterparts, with the purpose of assessing online performances. A measure of comparison of performance for online algorithms is the *competitive ratio* [2]. A competitive ratio of $r > 0$ means that the performance of an online algorithm is at least a factor of $1/r$ of the performance achieved by the best offline algorithm. It is defined as the worst-case ratio between its cost and that of a hypothetical offline algorithm which knows the entire sequence of requests in advance and chooses its actions optimally. Frederiksen [2] indicates that any reasonable deterministic or uniform randomized algorithm for packet transmission has a competitive ratio of exactly 2, where an algorithm is called reasonable if it does not postpone the transmission of a message by more than the sum of the inter-packet gap and overhead values.

> ***Total Latency Cost and Negative Slack***: Given a sequence $\sigma = \{PDU_i\}_{i=1,\ldots,n}$ of PDUs, where each packet i is released and stamped at time $Tstamp_i$, and arrives at the final destination at time $Tarr_i$, then the cost function for the total

latency of the PDU travel time is:

$$C_{Ttrav}(\sigma) = \sum_{i=1}^{n} (Tarr_i - Tstamp_i) \qquad (1)$$

Similarly, if the simulator is reading PDUs from a summary log file already captured, and each PDU_i is read for the first time at simulation time $Tread_i$, the cost function that measures the absolute value of the total negative slack time is:

$$C_{Tslack}(\sigma) = \sum_{i=1}^{n} (Tstamp_i - Tread_i) \times H(Tread_i - Tstamp_i) \qquad (2)$$

where H represents the Heaviside step function used to select only the negative slack occurrences:

$$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \qquad (3)$$

2.1 Online Bundling Strategies

The proposed online algorithms try to identify compatible PDUs P_1 and P_2 that can be bundled in a block $B = P_1 \otimes P_2$. A requirement is that the two PDUs should have the same type and length. Yet, because P_2 has not arrived by the time P_1 is being processed, the generating site must decide whether it will send P_1 immediately, or will wait for P_2. Chances are that P_2 will not be compatible with P_1. So if the generator could predict the type and the length of P_2, then the prediction could be used in the decision process. It is more difficult to predict both, type and length, than only one variable. For decisions based on one variable only, `type` is preferred due to its better discrimination among PDUs.

2.2 *Always-Wait, Always-Send* and *Neural-Network* Online Predictions

Two straightforward online algorithms are proposed to predict the type of the next PDU: *Always-Wait* and *Always-Send*. The former predicts that the next type will be the same as the type of the current PDU and takes the decision of waiting, using a timeout of 100 milliseconds. The latter predicts a type different from the current PDU type and never waits. It sends the PDU as soon as the time gap elapses without bundling it. A third online method is based on a Neural Network (NN)used to predict the PDU type[12]. A neural network was trained to predict the type of the next PDU based on the recent history of past PDUs. The NN architecture contained 44 input nodes, 20 hidden nodes, and 5 output nodes that specify the predicted type using a binary representation. The population of PDUs was split in two equal size sets for training and prediction. The program run for several days, and after 11,930 epochs, the percentage of successful predictions reached almost 70 %.

2.3 Offline Predictions Based on Type, Length, and Timestamp

The first offline prediction considers the PDU type only and is called *Type*. If two consecutive PDUs are of the same type, this method predicts *Waiting*, otherwise it predicts *Sending*. The second method *Type-Length* considers the type and the length of the PDUs. *Waiting* is predicted if two consecutive PDUs agree in both, type and length. The last method requires the same type, length, and timestamp for consecutive PDUs and will be called *Type-Length-Time*.

3. PACKET ALLOYING

Compression and aggregation of network packets are techniques used to reduce the total traffic, requiring less bandwidth for sending data. During the bundling process, decisions have to be made about whether to bundle or not consecutive packets. Considering that after sending one packet, a certain minimum time gap must elapse before sending the next packet, then the decision is not trivial. If the first packet is sent immediately, the second one could be delayed more than if the two packets are sent in a single bundle. This decisions are referred to as the *Packet Bundling Problem* [2]. In the case of OTB simulations, the offline bundling is carried out based the type and length of the PDUs from the log files, estimating the time gaps.

Several characteristics of the OTB traffic are considered for bundling purposes. The discussion here applies not only to PDUs generated under the DIS protocol, but also to packets from other protocols as well. During the simulation, the participants interact with each other in real time. For example, if one simulated vehicle starts moving, or decelerates, all the other entities should be informed of the event as soon as possible. This characteristic impacts bundling in several aspects. First, the time a PDU waits for the upcoming PDU creates a delay against the meaning of real time. Therefore, a small timeout should be introduced to limit that waiting time. Second, not bundling PDUs could cause that the following PDU will be delayed even more, due to the gap time that must separate frames. Also, not bundling produces more traffic and longer router queues, which finally goes in detriment of the real time properties.

The decision of bundling PDUs must consider the pros and cons of each alternative. It is possible that for some environments, bundling is not a necessity. For example, a scenario in which a few sites are simulating a simple vignette connected in a LAN not requiring a router, could have enough bandwidth to manage the traffic without bundling. Some high priority PDUs like fire and detonation occur in short bursts, and they are usually sent at the same time, creating bottlenecks or negative slack spikes that attempt against the real time approach. Bursts of PDUs timestamped at the same or almost the same time encourages benefits of bundling, because not doing it causes a large negative slack spike at the transmitting site that delays the following PDUs. Essentially, instantaneous transmissions overwhelm the channel capacity so that the available bandwidth appears low relative to the demand. If the bundling operation is not time-consuming and the waiting timeout is selected appropriately, bundling a sequence of consecutive PDU and sending a single block could take less time than sending the individual PDUs without bundling them. For instance, if a PDU is 512 bytes long and the bandwidth is 64 Kbps, the transmission time of one single PDU is 64 milliseconds, while bundling several PDUs could take less than one millisecond. It is well documented that Embedded Simulation (ES) traffic contains 70 % or more of ES-PDUs. Usually these ES-PDUs are partially redundant, not urgent and less impacted by waiting to be bundled than other PDUs of higher priority. For example, if a vehicle is not moving, it still needs to send heartbeat ES-PDUs at regular intervals. Because they are so abundant, bundling and compressing ES-PDUs have a major impact on the overall traffic decrease. However, in this

research ES-PDUs were not bundled in the majority of cases because they did not participated in negative slack spikes as bursts of consecutive ones, as `po_fire_parameters` did.

PDUs may contain redundant field data, both inside each PDU and among PDUs. It has been observed PDUs from the OTB log files containing zeroes in the majority of their fields, with only two differences from one PDU to the next. Bundling and compression can take advantage of this high redundancy. *Packet Alloying*, the proposed algorithm, would append only the two differing fields in the second PDU to the first one. Other bundling algorithms also profit from redundancy, like those based on sending delta PDUs. Determination of the PDU structure based on its type and length allows a fast comparison among PDUs for algorithms like Packet Alloying. In DIS protocol, PDUs are broadcasted, thus simplifying the PDU header since a particular destination is not needed. Considering that all the PDUs in a bundle are delivered to the same recipients, bundling and routing processes become more straightforward. However, broadcasting contributes to the proliferation of messages sent to entities that might not need updated information from all the other entities, some of which could be very far away in the simulation field, and the non-reception of such messages is not going to significantly affect the simulation fidelity. Instead of broadcasting, multicasting is an alternative proposed by other protocols like HLA and DIS-Lite.

The slower the connections, the more significant the impact of bundling is. If a connection is slow, bundling and compression becomes more advantageous. In slow connections, gap times are larger, and so is the penalty for not bundling the next PDU. Also, a low bandwidth connection shows more negative slack times during transmission, causing bottlenecks and long queues. For instance, in the studied vignette the satellite connection introduces a propagation delay of about 0.25 seconds, much higher than the time gap required to separate frames during transmission. Therefore, bundling of high priority PDUs like `po_fire_parameters` tends to be worthwhile.

The slack time analysis at the ground station shows that many negative and positive slack spikes are present at regular time intervals. The size of such spikes depend on the timestamp and the length of the PDUs, and on the bandwidth of the channel. The following definitions and theorem formalize the occurrence and calculations of such spikes.

Definition 1 : *Busy Generator*.

> The generator of PDUs is in the *busy* state at time t if it is transmitting a packet or waiting for the completion of a time gap separator at that time.

Definition 2 : *Busy Phase*.

> Given a sequence PDU_1, ..., PDU_n of PDUs of lengths L_1, ..., L_n bits, respectively, and timestamped at ascending times T_1, ..., T_n seconds, respectively, then any subsequence PDU_i, PDU_{i+1}, ..., PDU_{i+k} constitutes a *busy phase* if the following conditions are true:
>
> 1. The generator is not busy at time T_i when PDU_i is released.
> 2. For $j = 1, 2, \ldots, k$, when PDU_{i+j} is released at time T_{i+j} the generator is busy completing the transmission of previous PDUs.
> 3. If PDU_{i+k+1} exists, the generator is not busy at time T_{i+k+1}.

Definition 2 indicates that the entire sequence of PDUs can be partitioned into disjoint phases of consecutive PDUs, and each PDU belongs to one and only one phase. The following theorem calculates the size of negative slack spikes during a busy phase. A formal proof is found in [10].

Theorem 3.1

> The magnitude of a negative slack spike of a busy phase PDU_i, PDU_{i+1}, ..., PDU_{i+k} transmitted at a bandwidth B bps with gap intervals of g seconds can be calculated as:

$$m = \max_{0 \le j \le k} \{m_{i+j}\} \tag{4}$$

$$\text{where } m_{i+j} = T_{i+j} - \left(T_i + j \cdot g + \frac{\sum_{u=0}^{j-1} L_{i+u}}{B} \right)$$

If a busy phase contains only one PDU, then Theorem 3.1 yields zero for the magnitude of the spike, and yields a strictly positive value for two or more PDUs, accepting by convention that the summation from index 0 to index -1 is zero. In other words, phases consisting of only one PDU produce no negative spike, and phases of two or more PDUs can produce a negative spike.

Positive spikes are always produced at the end of a phase, provided that the next phase does not start exactly at the end of the previous phase, including the gap separators as part of the phase time. In other words, positive spikes are produced by the time interval separating busy phases. The following theorem formalizes the concept.

Theorem 3.2

> The magnitude of a positive slack spike between consecutive busy phases (PDU_i, ..., PDU_{i+k}) and (PDU_{i+k+1}, ..., PDU_{i+k+r}) transmitted at bandwidth B bps with gap intervals of g seconds is calculated as:

$$m = T_{i+k+1} - \left(T_i + (k+1) \cdot g + \frac{\sum_{u=0}^{k} L_{i+u}}{B} \right) \tag{5}$$

3.1 Minimum Bandwidth Requirements

An independent analysis of traffic can be carried out without employing simulation by merging the PDUs of all the sites in one single stream of data sorted according to their timestamps. Then, the minimum bandwidth requirements can be estimated by selecting a small time interval (i.e 2 seconds) and calculating the average bandwidth for that interval. The time interval requiring the maximum average bandwidth is a good estimate of the minimum bandwidth required for the entire simulation. Although no overheads like

retransmissions, packet losses, or collisions are considered in the calculation of the bandwidth, a time gap separation between consecutive PDUs was included in accordance with the IEEE Std. 802.11 Formally, the definition of minimum local bandwidth and related concepts follow.

Definition 3 : ***Minimum Local Bandwidth.***

> Let all the PDUs in the simulation be sorted in ascending order of timestamp and numbered PDU_1, ..., PDU_n. Let L_i and T_i represent the length in bytes and the timestamp in seconds of PDU_i for all $1 \leq i \leq n$. Let g denote the minimum separating time gap between PDUs in seconds. If $i < j$, the *minimum local bandwidth* for the time interval $[T_i, T_j)$ is the minimum bandwidth in the output channel such that all the consecutive PDUs and gaps $[\text{PDU}_i, g, \ldots, \text{PDU}_{j-1}, g]$ can be successfully transmitted during this interval on or after the times T_k, respectively, for $i \leq k \leq j-1$.

According to Definition 3, it is allowable to transmit PDU_k on or after the time T_k, provided that at time T_j when the interval has elapsed, all the preceding PDUs have been transmitted. It should be noted that the time interval $[T_i - T_j)$ includes PDU_i and does not include PDU_j. Definition 3 assumes that the timestamps are different for consecutive PDUs, such that $T_i \neq T_j$ for all $i \neq j$. If consecutive PDUs bear the same timestamp, the minimum local bandwidth for the corresponding time interval would be infinite, and the PDU sequence is said to be *not feasible*. If the time interval tends to be small relative to the simulation time, then the minimum local bandwidth approaches minimum instantaneous bandwidth, as it is in the extreme case that $j = i + 1$ and only one PDU lies in the interval.

Definition 4 : ***Minimum Instantaneous Bandwidth.***

> The minimum local bandwidth is called *minimum instantaneous bandwidth* if $j = i + 1$ such that the time interval over which it is calculated contains a single PDU.

In practice, any time interval considered short within the context of the simulation time can lead to the concept of minimum instantaneous bandwidth. The successful transmission of PDUs mentioned in definition 3 depends on both the length and the timestamp of all the PDUs. The minimum average bandwidth defined next is an approximation to the minimum local bandwidth that in practice gives sufficiently precise results. This measure was used in the architecture-independent analysis of the OTB vignettes.

Definition 5 : ***Minimum Average Bandwidth.***

> Under the same notation and conditions of the definition of minimum local bandwidth, but disregarding the intermediate timestamps T_{i+1}, ..., T_{j-1}, the *minimum average bandwidth* is the minimum bandwidth in the output channel such that all the bit volume plus all the gaps separating participating PDUs can be transmitted during the time interval $[T_i, T_j)$.

In calculating the minimum average bandwidth, it is convenient to select a fixed size of S seconds for the time intervals, and divide the total simulation time into subintervals of this fixed size. In such a case, the above definitions can be applied to the time intervals $[T_i, T_j)$, provided that T_i and T_j are separated by a time distance of at least S seconds, but T_i and T_{j-1} are not. In other words, the time interval $[T_i, T_j)$ is of minimum length not less than S seconds, for a given constant S selected a-priori. In the independent analysis of the said vignettes, S was chosen equal to 2 seconds, and so the calculated bandwidth was considered instantaneous.

Theorem 3.3

> Under the same notation and conditions of the definition of minimum local bandwidth, and assuming the following condition of feasibility:
>
> $$T_j - T_k - (j-k)g \quad > 0, \quad \forall\, k:\ i \leq k < j \quad (6)$$
>
> then, the minimum average bandwidth $\overline{B}_{i,j}$ in bits/second required to transmit the PDUs PDU_i, ..., PDU_{j-1} during the time subinterval $[T_i, T_j)$ is calculated as:
>
> $$\overline{B}_{i,j} = \frac{\sum_{k=i}^{j-1} 8L_k}{T_j - T_i - (j-i)g} \quad (7)$$

The condition of feasibility in Equation 6 says that the remaining time from T_i to T_j should be sufficient to accommodate all the gaps between PDUs and still have capacity for the transmission of content bytes. The average bandwidth for the interval $[T_i - T_b)$ is the ratio of the total number of bits transmitted over the remaining time in the interval once the gaps have been deducted, as stated in Equation 7.

Theorem 3.4

> Under the same notation and conditions of the definition of minimum local bandwidth, and assuming the condition of feasibility given in theorem 3.3, if $a < b$ and PDU_a, ..., PDU_b are consecutive PDUs in the sequence, then the minimum local bandwidth $B_{a,b}$ for the interval $[T_a, T_b)$ is given by:
>
> $$B_{a,b} = \max_{a \leq k < b} \{\overline{B}_{k,b}\} \quad (8)$$

According to Theorem 3.4, if the bandwidth for the whole interval $[T_a, T_b)$ is constant, it should not be less than any individual average bandwidth $\overline{B}_{i,b}$. Considering that the starting time for transmitting PDUs can be delayed within the interval, Equation 8 takes the maximum of all components $\overline{B}_{i,b}$ representing the minimum bandwidth requirement.

3.2 Development of Alloying in Simulation Model

After analyzing all of the PDUs in the log file for a given vignette, it was observed that the type and the size of a PDU can adequately determine its internal field structure. In order to better explain the different flavors of bundling algorithms, we need to formalize some definitions and propose a conjecture.

Definition 6 : *Compatible PDUs*.

> Two PDUs $A = (a_1, a_2, a_3, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ are said to be *compatible* if and only if $type(A) = type(B)$ and $length(A) = length(B)$, assuming that *type* and *length* are functions that return the type and the length in bytes of a PDU, respectively.

Conjecture 3.1

> If the PDUs $A = (a_1, a_2, a_3, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ are compatible, then $n = m$ and $field\_type(a_i) = field\_type(b_i)$ for all $1 \leq i \leq n$, assuming that *field_type* is a function that returns the type of any field in the PDU.

Conjecture 3.1 cannot be proved unless the formal specifications of PO_PDUs are analyzed and the OTB source code is examined, items not made available to this research. However, all of the 60,341 PDUs in the main vignette were checked, as well as the PDUs in other two vignettes, and no exceptions to Conjecture 3.1 were found, which empirically prove the assertion. Because compatible PDUs share a common internal structure, they are good candidates to be bundled. The other requirements to deliver the PDUs as a single packet are that they must be consecutive and scheduled within a short time interval. The pseudo-algorithm of PDU bundling is described in Figure 2.

If several PDUs are scheduled at the same or almost the same time, and the structure of those PDUs is the same, with only small but predictable differences, then only one single PDU needs to be sent along with instructions on how to recover the other PDUs from the given one. Comparisons of `po_fire_parameters` type of PDUs among other PDUs involved in the same negative spike showed that the stated conditions (same timestamp, small differences) can be exploited. These PDUs differ on consecutive identification attributes (like counters), and memory addresses that change according to the PDU length. The bundling method called *Packet Alloying* was proposed after analyzing several PDUs captured in the log files. In those logs there were cases of consecutive PDUs almost identical, with zeros in many fields. In all of the negative spikes studied, compatible PDUs had similar redundancies. Extraction is the inverse procedure of alloying. The formal definition of *Packet Alloying* follows, which is taken from [12] and is the basis for the bundling algorithm proposed in Section 3.2.

Definition 7 : *Packet Alloying*.

> Let $N = \{1, 2, \ldots, n\}$ be a set of indices, and let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ be two consecutive PDUs, where A and B are of the same type and the a_i and b_i represent PDU fields. For a subset $S \subseteq N$ such that $a_i = b_i$ for all $i \in S$, the bundle of A and B is defined as the PDU $A \otimes B = (a_1, a_2, \ldots, a_n, [(b_j, j)_{j \in N \setminus S}])$. A is called the *reference* PDU in the bundle.

In the above definition, the notation $N \setminus S$ represents the set difference of N and S. The square brackets [,] denote the start and end, respectively, of replicated PDUs to be formed during extraction. The definition can be extended to any number of PDUs. The said bundle is called *Packet Alloying* because it is formed like a metal alloy, bundling PDUs based on their internal structure.

```
1. Wait until (next PDU is ready for delivery
     Let A denote that PDU;
2. Block = A;
   /*This is the first PDU in the bundle*/
3. Set timeout = maximum time PDU A
   will wait in Block;
4. While (timeout not expired)
5.  {If (next PDU is ready for delivery)
         Let B be that PDU;
     else Continue at the While loop;
6.   If (A and B are compatible PDUs)
         /* Packet Alloying bundle*/
         {Block = Block (x) B;
          B = empty;
         }

     else break the While loop;
    } /* End While */
7.  Send Block as a single packet;
8.  If (B is empty)
          Continue at Step 1;
    else {A = B;
          Continue at Step 2;
         }
```

Figure 2: Pseudo-Algorithm of PDU Bundling. Operator (x) in Step 6 represents the Packet Alloying operation

A performance issue deals with the convenience of bundling compatible PDUs in all situations. According to Definition 7, each field in the new PDU that differs from the corresponding field in the reference PDU is appended to the bundle along with an index. In the extreme case that all the fields in the new PDU are different from the corresponding fields in the reference PDU, the new PDU will be bundled with no reduction in size due to a lack of redundancy elimination. Moreover, the inclusion of indices in this case would make the bundle larger than the sum of the sizes in the two individual PDUs. A threshold parameter could be used for comparing the size of the bundled fields and indices to the size of the unbundled PDUs and decide whether the bundle is worth or not.

If a bundling operation is performed, some time should be spent in the bundle operation itself. In other words, it is not possible to process a PDU (read and bundle it) in zero time. It was estimated that each bundle operation requires 5 microseconds of simulation time. This time could be considered as generator *service time*. The generator busy time is then computed as the transmission time, plus the gap time, plus any service time if a bundling operation is carried out.

4. RESULTS OF THE SIMULATION

The analysis of negative spikes motivated the concept of a possible solution to eliminate or reduce those spikes by means of aggregating the participating PDUs. An analysis of all the logged PDUs revealed that if two PDUs are

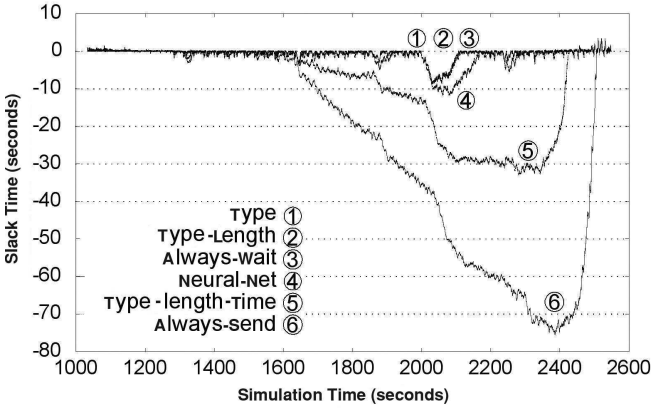

Figure 3: Slack Time at Ground Station for the 6 Predictive Strategies (64 Kbps).

of the same type and length then they have identical field structure. This observation constitutes a key point in the proposed bundling algorithm. Logged PDUs also revealed that OTB schedules some sequences of consecutive PDUs at exactly the same timestamp, which causes a bottleneck in generators due to the infeasibility of sending several packets at the same time. In most cases, consecutive PDUs of equal type and length differed in the contents of only a few fields, presenting the possibility of merging them.

4.1 Slack Time Analysis

Figure 3 shows the slack time of the generator at the CONUS ground station for 6 different predictive algorithms. The graph was created assigning 64 Kbps to all the wireless links and 100 milliseconds to the timeout period. As seen in the diagram, up to the second 1,600, all of the algorithms behaved alike, but afterwards negative slack started to build up. The *Always-Send* algorithm, equivalent to the *non-bundling* option, incurred in the largest negative slack, followed by the *Type-Length-Time* strategy. The NN approach performed relatively well, considering that its predictions are not perfectly accurate. The other 4 algorithms are among the best in this simulation, and a closer examination reveals that the NN approach could be improved. NN predicts the PDU type based only on the time series of the past 44 PDUs. Comparing its performance against the optimal *Type* algorithm, its competitive ratio for the cost function *negative slack time* was $c = 3.75$.

It was also observed that the decision of sending the current bundle based solely on the upcoming PDU type, performs as well as the one that considers the type and the length of each PDU. Therefore, a neural network approach could benefit from this fact by concentrating the effort in predicting the type only, instead of the type and the length.

However, the most interesting observation is that the simplest *Always-Wait* is almost as good as *Type-Length*. The reason is that there is a high probability that the prediction based solely on the type agrees with the prediction based on the type and length. For example, an offline examination of the PDUs indicated that from the 50,230 PDUs sent by the CONUS ground station, 42,911 (85.4 %) implied the same action (wait or send) for both algorithms.

Table 1 shows the slack time average and standard deviation for all combinations of algorithms and bandwidths measured at the ground station. The average is a signed number calculated from all the PDUs generated during the simulation.

Table 1: Slack Time Average and Standard Deviation for All the Studied Algorithms and Bandwidth Combinations Measured at the Ground Station. Best offline and online values are underlined.

Average / Std. Deviation	64 Kbps	128 Kbps	256 Kbps	512 Kbps
Type	-0.758	-0.017	0.015	0.024
	1.600	0.109	0.073	0.066
Type-Length	-0.760	-0.018	0.015	0.024
	1.601	0.110	0.073	0.066
Type-Length-Timestamp	-10.659	-0.027	0.013	0.023
	11.711	0.115	0.073	0.066
Always-Wait	-0.802	-0.017	0.016	0.024
	1.689	0.109	0.073	0.066
Neural-Network	-1.579	-0.044	0.008	0.022
	2.638	0.162	0.085	0.069
Always-Send	-26.181	-0.054	0.006	0.021
	26.033	0.176	0.085	0.069

From Table 1 it is concluded that *Always-Send* is the worst of the 6 algorithms, and *Always-Wait* is among the best ones. Because, *Always-Send* corresponds to the non-bundling option, it is clear that the proposed bundling is advantageous when compared to DIS protocol.

Also, at 64 and 128 Kbps, the average slack time was negative in all cases. A negative average indicates that the corresponding bandwidth is insufficient to handle the PDU traffic. Therefore, for this vignette the wireless bandwidth should be at least 256 Kbps.

4.2 Travel Time Analysis

To enable analysis, each sent bundle includes the current time (T_{send}) attached to it, allowing the destinations to calculate the travel time T_{trav}. Figure 4 shows the travel time of *Always-Wait* measured at a given site onboard one of the planes, using 64 Kbps, and latter 128 Kbps in wireless links. It is clear from the graph that 64 Kbps is not enough to handle the required traffic, even with bundling. As seen, during the interval from second 2000 to second 2400, many of the PDUs took almost 40 seconds to arrive at their destinations, exceeding the fidelity requirements of the OTB simulation. However, a big improvement is obtained just by duplicating the bandwidth. At 128 Kbps, the latency was close to 0.8 seconds, and most of the PDUs took less than 0.4 seconds to reach their destinations. There was a large concentration of PDUs taking near 0.25 seconds of travel time, corresponding to the propagation delay of satellite signals. However, some PDUs took less than 0.1 seconds of travel time, corresponding to messages sent from other airplanes not using the satellite.

Table 2 shows the average and standard deviation of the travel time for each combination of algorithm and bandwidth, measured at sink 0 onboard plane 0. Considering that approximately 83 % of the PDU traffic arriving at sink 0 comes from the ground station via satellite, and that for those PDUs, 0.255 seconds is an unavoidable delay, the table

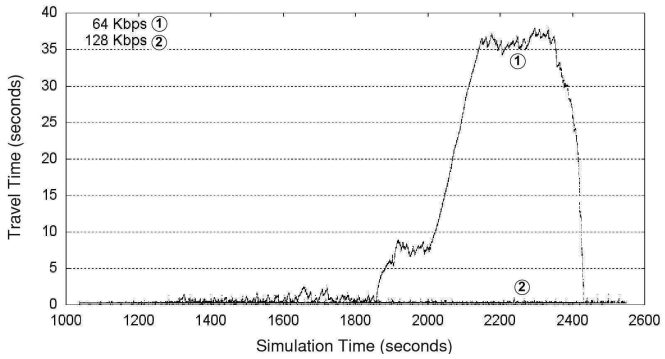

Figure 4: Travel Time for the *Always-Wait* strategy, at 64 Kbps and 128 Kbps.

shows a very good behavior of the algorithms at 256 Kbps or more, giving a slight advantage to *Always-Wait* and *Neural-Network* over *Always-Send*.

Table 2: Average and Standard Deviation of Travel Time Measured at Sink 0

Avgerage / Std. Deviation	64 Kbps	128 Kbps	256 Kbps	512 Kbps
Type	9.20	0.304	0.262	0.249
	13.2	0.099	0.069	0.064
Type-Length	9.24	0.306	0.262	0.249
	13.2	0.101	0.069	0.064
Always-Wait	9.43	0.303	0.261	0.249
	13.5	0.099	0.069	0.064
Neural-Network	28.7	0.314	0.261	0.248
	33.2	0.119	0.069	0.064
Always-Send	64.0	0.333	0.263	0.251
	58.0	0.153	0.062	0.057

Table 3 shows the total travel time for all the PDU bundles that arrived at a particular node in an aircraft. The sum of travel times over all the bundles can be a possible cost function used to estimate competitiveness of the online algorithms. According to Table 3, at 64 Kbps the best offline algorithm is *Type-Length*. Based on it, *Neural-Network* would have $c = 3.75$ and *Always-Wait* would have $c = 1.03$. However, we cannot assume that *Type-Length* is the optimal offline algorithm. In fact, *Type-Length* can be improved as follows. After processing a given PDU, if *Type-Length* predicts `W` (*wait*), but the next PDU arrives after the timeout of the current bundle, then the waiting time was wasted. A better offline algorithm could have analyzed this case and predicted `S` (*send*).

At 256 Kbps, *Type* appears better than *Type-Length*, and the online algorithm *Always-Wait* seems to win. This information is contradictory, and we explain it by saying that there is a better offline algorithm that overcomes those in Table 3. Nevertheless, a conclusion drawn is that, at higher bandwidths, differences among the algorithms become smaller. For instance, at 256 Kbps *Neural-Network* has $c = 1.15$ based on *Type-Length*, instead of $c = 3.75$.

For transmissions clearly exceeding the channel capacity, *Type* is the best choice, resulting in a 89.3 % improvement

Table 3: Total Travel time at sink 21 (64 Kbps, 256 Kbps)

Bandwidth	Strategy	Total Travel Time (seconds)
64 Kbps	Type	222,589.264
	Type-Length	222,357.200
	Always-Wait	228,350.418
	Neural-Network	832,881.794
256 Kbps	Type	8,419.056
	Type-Length	8,431.477
	Always-Wait	8,357.483
	Neural-Network	9,725.795

compared to *Always-Send* used by DIS. However, if the channel capacity is near to the demanded rate, then *Always-Wait* can perform just as well, yielding a 30.3 % improvement over DIS. On the other hand, for a low bandwidth a *Type* strategy can outperform *Always-Wait* by 3.1 %. These results are not surprising because *Type* is offline, and good offline algorithms should outperform the online ones.

Among the studied online algorithms, the closest one to *Type* is *Neural-Network* that strives to predict the type of the next PDU in the sequence. Assuming that *Neural-Network* could be improved sufficiently to resemble the performance of *Type*, and defining the coefficient γ to be the ratio of the channel capacity to the average bandwidth demand:

$$\gamma = \frac{channel\ capacity}{average\ bandwidth\ demand} \tag{9}$$

then γ can be used to select the preferred PDU bundling strategy. For low values of γ ($\gamma < 1$)the demanded bandwidth is larger than the channel capacity. *Type* is the best offline algorithm in this case, but due to its offline nature, an improved *Neural-Network* is selected. If γ is somewhat larger than 1, for instance between 1 and 2 $1 < \gamma < 2$, the channel capacity is sufficient to handle the traffic on the average, but there could be spikes of high demand, and *Always-Wait* is the best choice in this scenario. When γ is large, for instance larger than 2 ($\gamma > 2$), there is an excess of bandwidth as compared to the demand, and alloying is not justified. Alloying implies the addition of a small delay while the algorithm is waiting for the next PDU. *Always-Send* is a good choice in this situation because it is simple, does not incur in extra delays and provides good performance.

4.3 Queue Length Analysis

Due to the nature of the PDU traffic in the simulation, two queues to focus attention on are the router queue onboard any of the aircrafts and the satellite queue. Figure 5 shows the satellite queue at 64 Kbps and 128 Kbps. It is clear from the graph that 64 Kbps is an insufficient bandwidth, causing the satellite queue to grow unbounded once it becomes full. The reason for having a descent after reaching a maximum of about 6,000 messages, is that the simulation is approaching its end and no more messages are sent from generators. However, at 128 Kbps a significant change in the queue length is produced, keeping it at reasonably low values.

Also, at 64 Kbps the graph does not reach zero at the

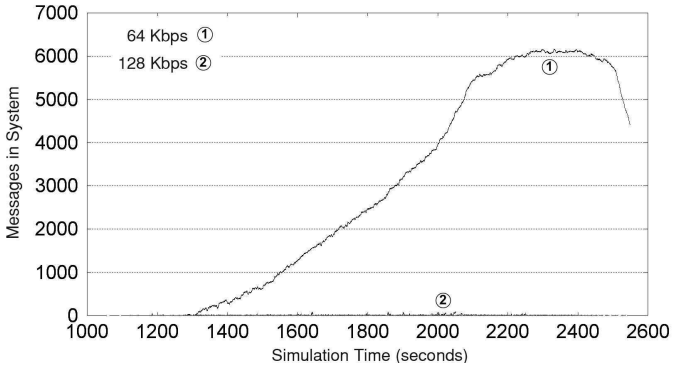

Figure 5: Messages in Satellite Showing the Impact of a Higher Bandwidth on its Queue (64 Kbps and 128 Kbps).

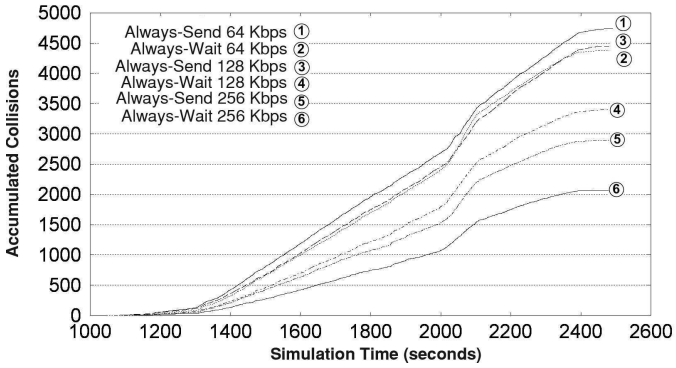

Figure 6: Collision Accumulation at Plane 7 (64, 256, 512, 1,024 Kbps).

end. This occurs because the queue status is reported only if another message enters the queue. After the arrival of the last message to the queue, messages are consumed without being reported.

Table 4 displays the average and standard deviation of the satellite queue length for combinations of different algorithms and bandwidths.

Table 4: Average and Standard Deviation in the Satellite Queue Length for Combinations of Algorithm and Bandwidth

Average: Std. Deviation	64 Kbps	128 Kbps	256 Kbps	512 Kbps
type	316.97	2.38	0.91	0.56
	411.43	3.97	1.72	1.23
Type-Length	318.154	2.44	0.92	0.56
	412.273	4.13	1.75	1.26
Always-Wait	327.278	2.30	0.85	0.49
	421.161	3.88	1.69	1.16
Neural	1,028.47	3.58	1.24	0.79
Network	1,045.26	6.37	2.18	1.52
Always-Send	2,962.94	5.40	1.22	0.63
	2,236.83	10.78	2.55	1.57

4.4 Collision Accumulation

Collision accumulation in one of the planes at different bandwidth rates is given in Figure 6. Results from the simulation indicate that at 64 Kbps the highest collision rate measured at the router aboard airplane 7 was close to 12 collisions per second, and it occurred during the time interval [2050, 2100] in the link connecting the satellite to the planes. At 64 Kbps, fewer than 4,800 collisions were detected in all for *Always-Send*, representing less than 8 % of the total number of PDUs. At 256 Kbps, collisions for *Always-Wait* were close to 2,100, or 5.3 % of all the bundles.

As Figure 6 shows, at 128 and 256 Kbps there is roughly a total difference of 1,000 fewer collisions for *Always-Wait* than for *Always-Send*, indicating that Packet Alloying significantly reduces collisions for the same bandwidth. Also, it can be noted that as the bandwidth increases, the number of collisions decreases, because at higher bandwidths PDUs take less transmission time, lowering the probability of a collision.

5. CONCLUSIONS

Several sets of simulations were performed, using different vignettes and different assignment of OTB sites to simulation nodes. The analysis of the slack time revealed the occurrence of negative slack spikes at regular time intervals. The studied PDUs participating in those negative spikes showed that they constituted sequences scheduled at the same or almost the same time, and usually of the same type and length. The structure of such PDUs was investigated, concluding that they were similar in structure. Results demonstrate that DIS traffic generated by OTB can be substantially reduced by the application of several techniques. PDU bundling techniques can diminish the negative spikes in the slack time during traffic generation. Inter and intra PDU redundancy can be eliminated by bundling and compression techniques.

Bundling PDUs contributes to save bandwidth by removing redundant fields, which is the base of the proposed Packet Alloying bundling. If several PDUs are produced, only one physical bundle is actually sent carrying the non-redundant fields of those PDUs. *Packet Alloying* is a lossless compression technique in which the extraction of the original PDUs occurs at the destinations in a straightforward manner. The new technique significantly lowered the queue lengths of routers and satellite, and decreased the travel times of PDUs at low bandwidths. Its performance was compared to that of the non-bundling option. The effect of bundling was significant, as indicated by the collected statistics. For instance, applying *Always-Wait* to the vignette at 64 Kbps in wireless links, a reduction in the magnitude of negative slack time from -75 to -9 seconds (88 %) for the worst spike was achieved. Similarly, at 64 Kbps, *Always-Wait* reduced the average satellite queue length from 2,963 to 327 messages for a 89 % reduction. Although their performance can vary, all the algorithms utilizing the proposed bundling strategies performed significantly better than the non-bundling *Always-Send* algorithm. By applying *Packet Alloying*, it was found that 256 Kbps in wireless channels is the minimum bandwidth for the studied vignette that passes the test by all metrics (slack time, travel time, queue length,

and collisions).

Three online algorithms were proposed: *Neural-Network*, *Always-Wait* and *Always-Send*, as well as three offline algorithms: *Type*, *Type-Length* and *Type-Length-Time*. The *competitive ratio* was estimated for the online algorithms compared to the offline ones. For *Always-Wait* the index was very close to 1, and we conclude that those offline algorithms are not optimal. Also, the NN strategy can be improved, possibly by using more neurons, a longer input sequence, and extended training sessions. Data revealed that predictions based solely on the PDU type are almost as good as predictions based on type and length, and these predictions are better than *Always-Wait* in many cases. Therefore, a NN approach could be useful if the percentage of successful guesses is high enough that it outperforms the *Type* or *Always-Wait* algorithms.

The *Always-Wait* algorithm, although not optimal, gives very good results especially at higher bandwidths. The simpler strategy *Always-Wait* performs as good as *Type-Length*, possible because there is a high probability that predictions based solely on the type agree with predictions based on type and length. For example, an offline examination of the PDUs indicated that from the 50,230 PDUs sent by the CONUS ground station, 42,911 (85.4 %) implied the same action (wait or send) for both algorithms. It was observed that at higher bandwidths, the difference between *Always-Wait* and the offline algorithms *Type* and *Type-Length* becomes smaller, giving more relevance to the straightforward *Always-Wait* strategy.

Even though the type of aggregation proposed proved to be successful for DIS transmissions, it can be applied to any protocol based on structured packets where field redundancy is observed, as is the case of TCP and UDP transmissions.

6. ACKNOWLEDGMENTS

I would like to thank Dr. Ronald DeMara who was my advisor in the Department of Electrical and Computer Engineering at the University of Central Florida (UCF) and chair of the Bandwidth and Latency Implications of Integrated Tactical and Training Communication Networks Project. Also, thanks to PEO STRI because this work was supported in part the U.S. Army Research Development and Engineering Command (RDE-COMM) as part of the Embedded Combined Arms Team Trainer and Mission Rehearsal (ECATT-MR) Science and Technology Objective (STO) contract N61339-02-C-0097.

7. REFERENCES

[1] L. M. Corporation. Advanced distributed simulation technology ii (adst ii) onesaf testbed baseline assessment (do #0069) cdrl ab02 final report. In *Proceedings of NAECON 88*, Dayton, Ohio, May 1998.

[2] J. S. Frederiksen and K. S. Larsen. Packet bundling. In M. Penttonen and E. M. Schmidt, editors, *Proceedings of the Algorithm Theory - SWAT 2002: 8th Scandinavian Workshop on Algorithm Theory*, volume 2368 of *Lecture Notes in Computer Science*, pages 328–337, Turku, Finland, July 3-5, 2002. Springer-Verlag Heidelberg.

[3] A. Goel, M. R. Henzinger, S. Plotkin, and E. Tardos. Scheduling data transfers in a network and the set scheduling problem. *Journal of Algorithms*, 48(2):314–332, 2003.

[4] IEEE Computer Society Press. *IEEE Std 1278-1993, 1278.1-1995, 1278.2-1995, 1278.3-1996, 1278.1a-1998, IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulations Applications. Entity Information and Interaction, Application Protocols, Communication Services and Profiles, Recommended Practice for Distributed Interactive Simulation - Exercise Management and Feedback*, 1993, 1995, 1996, 1998.

[5] R. M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92*, volume 1, pages 416–429, Madrid, Spain, September 7-11, 1992. North-Holland.

[6] B. McDonald, J. Weeks, and J. Hughes. Development of computer generated forces for air force security forces distributed mission training. In *Proceedings of the 2001 I/ITSEC*, Orlando, FL, U.S.A., November 26-29, 2001.

[7] S. Phillips and J. Westbrook. On-line algorithms: Competitive analysis and beyond. In *Algorithms and Theory of Computation Handbook,*. CRC Press, 1999.

[8] D. Taylor. Dis-lite & query protocol: Message structures. In *Proceedings of the 14th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL, U.S.A., March 11-15, 1996.

[9] A. Varga. Omnet++ discrete event simulation system, version 2.3, user manual, June 15 2003.

[10] J. J. Vargas. *Data Transmission Scheduling for Distributed Simulation Using Packet Alloying.* Dissertation, University of Central Florida, Department of Electrical and Computer Engineering, Orlando, Florida, U.S.A., December 2004.

[11] J. J. Vargas, R. DeMara, A. Gonzalez, and M. Georgiopoulos. Bandwidth analysis of a simulated computer network running otb. In *Proceedings of the Second Swedish-American Workshop on Modeling and Simulation (SAWMAS 2004)*, Cocoa Beach, FL, U.S.A., February 2004.

[12] J. J. Vargas, R. F. DeMara, M. Georgiopoulos, A. J. Gonzalez, and H. Marshall. Pdu bundling and replication for reduction of distributed simulation communication traffic. *JDMS: The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1(3):173–185, 2004.

[13] J. J. Vargas, F. Goergen, R. DeMara, A. Gonzalez, and M. Georgiopoulos. Interim report: Bandwidth and latency implications of integrated tactical and training communication networks. Technical report, University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL, U.S.A., July 6, 2003.