Design and Implementation of Socket-level Bandwidth Aggregation Mechanism for Mobile Networking Environments

Hiroshi Sakakibara,† Masato Saito† and Hideyuki Tokuda††

This paper presents a mechanism that aggregates the bandwidths of multiple network interfaces on computers in mobile environments. Research and development on wireless technologies has recently been extremely active. As a result, various kinds of wireless media have been emerging and these have been installed on computers. However, the demand for broader bandwidths has not yet been met. We designed and developed a socket-level bandwidth aggregation mechanism (SBAM) that offer broader bandwidths using multiple wireless interfaces on computers. The most distinctive feature of SBAM is that the mechanism is implemented as a feature of operating systems. This involves two benefits. First, it is possible to deploy SBAM gradually in computers on the Internet. If the bandwidth aggregation mechanism (BAM) is achieved in a network stack, it must be implemented in all computers on the Internet. However, it is not necessary to implement it in all computers on the Internet, since it can recognize its own existence on peer hosts and determine whether to use BAM or not. Second, it does not require modifications to most existing software. For example, if the mechanism is achieved as a feature of a transport protocol, such as TCP, all applications using TCP must be re-written to adapt to the new transport layer protocol. However, SBAM does not require this, since the socket hides the existence of the mechanism. We evaluated SBAM in an actual wireless network environment. We achieved an increase in throughput by a factor of 1.6. SBAM offers broader bandwidths for applications by utilizing various wireless devices on computers while it avoids modifications to existing environments.

1. Introduction

The number of small computers and PDAs using 802.11b, 802.11a, Infrared, Bluetooth, and Ultra WideBand media has increased due to the recent growth in network technologies. This has led to a situation where computers tend to have multiple wireless network interfaces. Furthermore, the volume of network services that consume broader bandwidths, such as VoIP and video streaming, has increased. Since these have tended to be created by assuming wired networks, bandwidth starvation has occurred because of narrower bandwidths. Even when multiple wireless network interfaces have been installed on computers, only one interface is used at a time. One solution to prevent this starvation is to exploit several interfaces simultaneously.

This paper proposes a system that can achieve a solution called the Socket-level Bandwidth Aggregation Mechanism (SBAM), which aggregates the bandwidths of several interfaces connected to computers, even if the media are heterogeneous. Since SBAM is implemented in the socket layer of the operating systems, it can avoid the drawbacks caused by implementation in network stacks or in applications, and achieves bandwidth aggregation efficiently with minimal changes having to be made to existing software. SBAM can also be exploited to adapt to network changes due to user mobility, i.e., the varying network environment as they move around. SBAM can be exploited to seamlessly adapt to network changes for applications.

Applications that require specific bandwidths, such as VoIP, benefit from SBAM. For example, although the Personal Handy-phone System (PHS) is widely used to connect to the Internet outdoors, its bandwidth is insufficient for VoIP applications. However if users have two different PHSs, they can exploit both PHSs using SBAM.

The rest of this paper is organized as follows: Section 2 presents problems that need to be addressed to achieve an aggregation mechanism in network stacks and their solutions. We introduce the system architecture for SBAM in Section 3 and its implementation in Section 4. Section 5 presents the results of evaluation. Finally we conclude and discuss the future work in Section 6.
2. Which Layer Should Implement Aggregation Mechanism?

Our aim in this research was to develop a Bandwidth Aggregation Mechanism (BAM) and deploy it practically in real life. To achieve this, BAM should be able to be exploited easily by various users who are unfamiliar with computer technologies. Three requirements thus need to be satisfied. First, BAM should be able to be deployed simply and gradually. Second, modifications to existing applications should not be required. Third, BAM needs to be able to deal with different kinds and numbers of wireless media so that it can be used by non-professionals. Corporations can prepare dedicated computers when BAM is leveraged in company situations. However, we assumed BAM would be used by computer non-professionals, and different kinds and numbers of wireless media would be exploited, e.g., houese utilizing 802.11b and Bluetooth.

The following summarizes these requirements:

A: Easy and gradual deployment
B: Modifications to existing applications unnecessary
C: Ability to handle different kinds and numbers of wireless media

Implementing BAM in network stacks seems appropriate, since it is network-related. However, this causes several problems with our requirements.

It is insufficient to implement BAM in link or network layers to achieve the requirement A. Implementing BAM in these layers requires the firmware or programs to be modified such as switches and routers in many computers. The 802.3ad\(^5\) standard involves a link-layer bandwidth aggregation mechanism between two computers with multiple network interfaces (N/Is). However, the bandwidth and delay of the links need to be exactly the same. In contrast, implementation in the transport layer requires a smaller number of computers to be modified compared with that in link or network layers. Only the end hosts need to be modified. However, deploying a new transport layer protocol takes a long time. For example, SCTP\(^6\) has recently been emerging. While it has brought a number of outstanding features with it, few applications utilize the protocol. It is inadequate to implement a BAM as an application or middleware to satisfy requirement B.

This has the same results as proposing a new transport layer protocol, which requires modifications to existing applications. While RCP\(^7\) and PTCP\(^8\) proposed well-designed transport layer protocols with BAM, they required applications to be modified to leverage these protocols. BAM needs available throughput and delay information between end hosts to efficiently utilize the resources of multiple N/Is to handle different kinds and numbers of wireless media, i.e., requirement C. Since link and network layers are processed based on hop-by-hop, BAM in these layers makes it difficult to acquire this information. As a result, achieving BAM in network stacks is not suitable for our purposes.

We focused on sockets as features of operating systems (OSes). When BAM is built into sockets, no modifications are required to existing applications. Since OSes have sockets, modifications to switches or routers are not required. In addition, when a peer host does not have BAM, the opposite peer host simply decides not to use BAM. This satisfies requirement A. Since sockets are features of an OSes, no modification are required to the existing applications, which satisfies requirement B. Furthermore, available throughput and delay information can easily be collected between end hosts. Therefore, we chose to implement BAM to extend socket functionalities.

This work was based on the previous work\(^9\) presented at a Japanese workshop, although it does describe more detailed evaluations. Furthermore, the results for throughput differ slightly with other the previous work. According to the Garg and Kappes\(^10\), the throughput for 802.11b was up to 6 Mbps. We re-evaluated their throughput, and obtained similar results.

3. SBAM

This section clarifies the system requirements, and then explains the architecture based on these. The functionalities, flow of data transmission, and reception are described last.

3.1 System Requirements

SBAM aims to utilize the several network interface cards (NICs) installed on computers to increase the available bandwidth.

We considered a source host (SRC) with multiple N/Is transmitting data to a destination host (DST) via the Internet. The SRC utilizes the N/Is simultaneously. First, the data to be transmitted must be divided into the number of N/Is and sent to each of them. The amount
of data must be determined based on the end-to-end link status between the SRC and DST. For example, since the bandwidths available to 802.11b and 802.11a differ, sending the same amount of data at the same time to each N/I will not effectively utilize the available bandwidth of the 802.11a link. Furthermore, the delays also differ due to the network status of the route. When the system transmits data without taking network status into account, none of the links can be exploited efficiently. The system and algorithm need to determine the appropriate amount of data to each N/I based on this link-status information. Furthermore, where one of the N/Is is no longer available, it should be detected and its data should be rescheduled to another N/I. When the DST has multiple N/Is, the SRC must be notified of their total number and all IP addresses.

Packets arriving at the DST must be aggregated and passed to applications. Since packets route through multiple paths, they do not arrive in sequence. Therefore, they need to be re-ordered.

Finally, recent wireless services, such as the Personal Handy-phone System (PHS) or 3G mobile phone networks, have incurred charges proportional to the amount of data or the duration of the connection. Users require a system that can restrict the use of these kinds of services. For example, the amount of data transmitted may be restricted to some thresholds during the day, while all the available bandwidth can be exploited at night. These thresholds need to be specified by the user.

The following functionalities are required as a whole:
- Dividing data according to number of N/Is.
- Deciding amount of data transmitted.
- Determining network status, e.g., delay and available bandwidth.
- Routing of packets amongst N/Is.
- Notifying system of user’s N/I-use intentions.
- Re ordering arrived packets.

The first five functionalities are for SRC, and the last one is for DST. SBAM aims to attain efficient use of multiple wireless devices by offering these functionalities.

### 3.2 SBAM Functions

SBAM consists of the six functions given in Fig. 1 to satisfy the requirements described in the previous section. All of these are implemented in the socket layer. Three of these, i.e., the Send-data Schedule Function (SSF), Send-data Division Function (SDF), and Receive-data Aggregation Function (RAF), are exploited by all connections. The remaining three work while the system is running.

**Policy Notification Function**
This function notifies of the user policy from the user to the kernel space, which are a users’ requests for N/I usage. Their unexpected use against user’s intentions is prevented by this notification.

**Network Monitoring Function**
This function collects the delay and available bandwidth for all links that are exploited by SSF and SDF. Delay is monitored by sending ICMP packets periodically to DST. This is smoothed as formula (1). \( RTT \) is the collected information and \( SRTT \) means smoothed \( RTT \). \( \alpha \) is the coefficient for smoothing. 0.9 is employed in our system, which is recommended and described in RFC793 [11] in detail.

\[
SRTT = \alpha SRTT + (1 - \alpha) \times RTT \\
(0 \leq \alpha \leq 1) \quad (1)
\]

The packet Pair method [12] is exploited in our system to collect information on the available bandwidth. Due to the space restrictions, we have not described the details of this method here.

In contrast to SSF, only one of either RAF, SDF, and the Network Monitoring Function (NMF) works on the system. If these functions are invoked for all connections, monitoring the same host increases the number of control pack-
ets with the increase in network connections. To avoid this situation, NMF holds the monitored data. When they are required by the other functions, the held data is passed to them if it has been monitored within a specified period.

**N/I status Notification Function**

This function notifies of the number of N/Is available on the DST, which enables SRC to send data to multiple N/Is on DST.

Imagine a situation where an SRC has one wired NIC and a DST has two wireless NICs. Here, SRC sends a transmission-request to DST. This request can be sent to either of the NICs on DST. DST sends back the number of NICs and IP addresses it has to SRC. SRC starts to transmit data to the NICs which DST notified it of. When the number of NICs changes, SRC is notified, and then SRC restarts the transmission.

**Send-data Schedule Function**

This function aimed to determine the most efficient amount of data to be transmitted by each link to utilize the network resources between SRC and DST. It calculates the optional amount of data to be transmitted to fill the bandwidth delay product (BDP) between SRC and DST. BDP is calculated from delay and the available bandwidth, which are acquired from NMF.

**Figure 2** outlines the flow for transmitted data. The two circular cylinders represent the two links between SRC and DST, and the volume of space represents BDP. This is calculated as \( d_i \times b_i \), where \( i \) is the number of links, \( d_i \) (sec) is delay, and \( b_i \) (bps) is the available bandwidth. BDP of each link is calculated before data are transmitted. BDP of link 1 is 50 and that of Link 2 is 100. The \( P_i \) packets are sent based on formula (2), where \( m_i \) is the Maximum Transfer Unit (MTU) and \( \alpha_i \) is the coefficient for each N/I use passed from the Policy Notification Function (PNF).

\[
P_i = \frac{\alpha_i b_i d_i - \min(\alpha_1 b_1 d_1, \ldots, \alpha_n b_n d_n)}{m_i}
\]

(0 \leq \alpha \leq 1, i = 1 \ldots n) \quad (2)

The formula is applied at the beginning of a connection. This aimed to equalize the amount of BDP in each link. This corresponds to the A of bottom cylinder in Fig. 2. The difference between the two links, 50 (= 100 − 50), is transmitted to Link 2.

The next step is to transmit data proportional to each link’s available bandwidth, which corresponds to the B in both cylinders in Fig. 2. The \( P_i \) in formula (3) is the number of packets sent to each N/I. \( m_i \), \( \text{gcd} \), and \( \text{lcm} \) corresponds to MTU, the greatest common divisor, and the least common multiple.

These two steps are intended to fill the multiple links between two hosts. To send data proportional to the bandwidth available for each link, the difference in BDP is first transmitted (formula (2)), then formula (3) is applied.

For example, consider the situation when there are two links, A and B. The available bandwidth, MTU, and delay of Link A are 2 Mbps, 500 bytes, and 100 ms, and that of B are 1 Mbps, 1,000 bytes, and 50 ms, respectively. At the beginning of connection, the sender transmits \((2,000,000 \text{ (bps)} \cdot 0.1 \text{ (sec)} - 1,000,000 \text{ (bps)} \cdot 0.05 \text{ (sec)})/(500 - 8) = 37.5 \equiv 38\) packets to Link A according to formula (2). Next, \((2,000,000 \cdot 1,000)/(500 \cdot 1,000,000) = 4 \) packets are sent to Link A, and one packet is sent to Link B one after the other according to formula (3). All BDPs are then filled and sufficiently used. If the real value of throughput is used for \( b_i \), \( P_i \) can be huge. To avoid this, the value is rounded off in implementation as follows. For example, imagine the bandwidth available for the two links are 180 bps and 680 bps. The rounded off values are 200 and 700, and the ratio is then 2 : 7. Furthermore, one of the link values is calculated to be one. The result is 1 : 3.5 \( \equiv 1 : 4 \). Finally, 200 and 800 are used as each link’s bandwidth.

This algorithm is based on the fact that the delay and available bandwidth are stable throughout the data transmission. When there are fluctuations in delay during transmission, it is impossible to fill the BDP between two hosts. This is because formula (2) eliminates the differences in delay. Since formula (3) adapts to the fluctuations in available bandwidth, it does
not cause this drawback.

Send-data Division Function
This function divides the data to be sent into packets according to the decision by SSF, and it appends an SBAM header.

SBAM header
An SBAM header consists of a four-byte number-of-sequences field and a four-byte data-length field. Figure 3 outlines the packet format for an SBAM header. The number-of-sequences field is exploited on DST to re-order received packets. The data-length field is exploited to distinguish whether the packet is an SBAM packet or not, and when it is, this indicates the length of its payload.

Receive-data Aggregation Function
This function aggregates and re-orders the packets, which were routed through multiple paths. Re-ordering exploits the number-of-sequences field in an SBAM header.

The behavior of re-ordering can be divided into two types depending on the lower transport-layer protocols. When a transport protocol is reliable, such as TCP or STP, the data are re-ordered correctly.

However, when the semantics of the transport protocol is like UDP, the situation is different. Since the semantics involves less delay and unreliability, the received packets cannot be re-ordered completely, which increases delay. The queue length in SBAM is fixed in such protocols to prevent delay from increasing. It is also prioritized more to pass data to applications faster than reliable transport protocols.

3.3 Data Transmission and Reception Operation
At the beginning of data transmission, NMF starts measuring the delay and available bandwidth for each link. If there are multiple N/Is on DST, information is sent through the N/I status Notification Function (N/I-NF) to inform to NMF. The amount of data to be sent from each N/I is then determined by SSF based on the link status, and the data are divided and the SBAM header is attached in SDF. Finally, these data are passed to a transport-layer protocol, such as TCP or UDP.

When the data are received, the N/I-NF notifies SRC of the number of available N/Is. The received data are re-ordered in sequence, the SBAM header is removed by RAF, and then it is passed to the application.

4. Prototype Implementation
We implemented parts of the SBAM functions in the FreeBSD kernel. We also implemented an Application-level Bandwidth Aggregation Mechanism (ABAM), to evaluate the algorithms in SSF. SBAM and ABAM were implemented in the environment listed in Table 1.

4.1 Kernel Implementation
The prototype SBAM implementation included SDF, SSF, and RAF. Figure 4 depicts these functions and their correlations inside the kernel then data is transmitted and received. The gray parts in the figure represent modified or newly created functions.

Data Transmission
A transport layer function is called from sosend() using struct protosw inet, in which the function pointers of the transport-layer protocols are stored, in normal operation for data transmission to implement FreeBSD. sosend()
passes the variables information for send() or write() system call to lower transport-layer protocols.

sosend() calls sbam_send() in SBAM instead of the transport-layer functions, in this case udp_send(), and this is called from sbam_send(). sbam_send() is a function with SDF and SSF. The data are divided into the number of N/Is and processed in flows different from this function. For example, consider sending data exploiting N/I 1 and N/I 2. When SBAM sends the data through N/I 1, sbam_send calls udp_send with N/I 1’s information. This is the same in the case of N/I 2.

Since the prototype SBAM does not have NMF, all available bandwidth and delay parameters leveraged in SSF are fixed at the same value.

Data Reception
The received data in normal operation is processed by ip_input() and udp_input() in sequence, then stored in the socket buffer. When the data are ready, waiting processes are awakened by sowakeup(), and scheduled to read data from the buffer.

When SBAM packets are received, sbam_input() is called by sowakeup() instead of awakening the waiting process. Received packets in sbam_input() are re-ordered to put them in sequence and the SBAM header is removed. The data are then stored in the socket buffer. This function contains RAF, details are in Section 3.2.

4.2 Application Implementation
To reiterate, ABAM is an application-level bandwidth aggregation mechanism, which contains most of the functionalities except for the Policy Notification Function (PNF). ABAM is implemented to evaluate SSF. When data transmission begins, the number of N/Is on DST is notified through N/I-NF. NMF them monitors the delay and bandwidth available between SRC and DST. After that, SRC sends data according to the formula given in Section 3.2.

Note that we exploited the Packet Train\(^{14}\) method for ABAM instead of the Packet Pair method used in SBAM. The Packet Pair method leverages two continuous ICMP packets. Since the DST processes these packets in the application layer, we could not acquire sufficient accuracy in the time gap between two acknowledgments. Packet Train, on the other hand, exploits more than two packets. ABAM actually exploits five packets of 1,400 bytes. As a result, we could acquire more accurate results for the available bandwidth.

Packets are re-ordered in sequence at DST using the packet sequence number in the SBAM header. This is then removed. Finally the data are passed to the applications.

5. Evaluation
We evaluated SBAM and its functionalities by exploiting ABAM. The items we evaluated are as follows:
- Comparison of SBAM with ABAM
- Effects of SSF
- Packet-arrival Sequence
- Overhead for SBAM Headers

5.1 Experimental Environments
We exploited two environments for the evaluation depicted in Figs. 5 and 6. The SRC in Fig. 5 has two IEEE 802.11b N/Is, and each is respectively connected to Networks 1 and 2 with wireless channels 2 and 11. The DST is connected to another network, Network 3, with a wired Ethernet. The SRC in Fig. 6 has two IEEE 802.11b N/Is, and each is respectively connected to Networks 1 and Network 2 with wireless channels 8 and 11. DST is connected to another network, Network 3, with a wired Ethernet.

The difference between the two environments is that the wireless router, A in Fig. 6, transfers data from Network 1. However, neither router...
Table 2  Throughput of SBAM and ABAM.

| Media Mode of Each N/I | Implementation | Throughput  |
|------------------------|----------------|------------|
| 11 Mbps + 11 Mbps      | SBAM           | 9.12 Mbps  |
| 11 Mbps + 11 Mbps      | ABAM           | 9.36 Mbps  |

in Fig. 5 does this. The radio channels for N/I 1 and N/I 2 in both experimental networks were set by hand so they did not overlap. If they is overlapped, it would have been impossible to transmit or receive data simultaneously due to radio interference.

5.2 Comparison of SBAM with ABAM

We compared the throughput and application codes of SBAM with those of ABAM.

Throughput

We conducted our experiments using network A with the following settings To evaluate the increase in throughput by exploiting two N/Is. We deployed SRC and DST that were installed with FreeBSD and SBAM. UDP was employed as the transport-layer protocol. To compare ABAM with SBAM, we deployed SRC and DST with native FreeBSD; ABAM was also installed on these hosts. Media data of 5.5 MB were simultaneously transmitted to Network 1 and Network 2. The throughput was measured by transmitting the data 50 times, and the results were averaged. Table 2 lists the averaged throughput.

The throughput for SBAM and ABAM increased by a factor of 1.6, in contrast with single N/I use. Therefore, the simultaneous use of multiple N/Is increases the available throughput. Why this throughput could not be twice as fast as that for single N/I use is not clear.

The throughput for SBAM was 2.6% lower than that for ABAM. SBAM may have had more overhead than ABAM, which was caused by exploring available N/Is, appending the SBAM headers, and checking out the network topology.

\texttt{sbam\_send()} searches the number of linked interfaces in single N/I by linear searching. \texttt{udp\_output()} is then called without most of the procedures being processed. \texttt{sbam\_send()} in dual N/Is searches the number of linked interfaces. SBAM then divides the data to be sent, appends an SBAM header, and re-writes the default gateway to send data through the intended N/I. At this time, \texttt{rtalloc()} is called twice to find out the default-gateway information. \texttt{rtalloc()} is a function to search the routing table.

We measured the execution times for \texttt{sbam\_send()} and \texttt{rtalloc()} with single and dual N/Is to clarify the overhead of \texttt{sbam\_send()} for two N/Is. We exploited the Read Time Stamp Counter (RDTSC) instruction for time measurements. The execution times for \texttt{sbam\_send()} and \texttt{rtalloc()} were measured by sending 1,400-byte data 50 times. The averaged results are in Fig. 7. The vertical axis represents the execution time for each function. The horizontal axis represents the number of N/Is. There is a 98.1 times overhead on dual N/Is compared with a single one. 47.1% of the overhead is caused by calling \texttt{rtalloc()}. Thus, implementing a binary search procedure to find a default gateway instead of exploiting \texttt{rtalloc()} will decrease the overhead and might improve throughput.

Comparison of Application Codes

In comparing the implementation of user land programs, ABAM required all the functionalities to be implemented. That meant using raw sockets, changing default gateways, and sending UDP datagrams. The program was 666 lines. The program run on FreeBSD with SBAM, on the other hand, could be written in 206 lines. This is merely a UDP socket programming. This implies it is easier for programmers to exploit SBAM than to achieve bandwidth aggregation in application programs.

5.3 Effects of SSF

We measured the effects of SSF, which were derived from formula (2) and (3). ABAM with this functionality was exploited for measurement. Delay and available-bandwidth information were given in advance, and MTU was fixed at 1,500 bytes. Furthermore, the coefficient for N/I, \(\alpha\), which is passed by users through PNF, was also fixed at one. The results are listed in Table 3.

When the available bandwidth was different, the throughput for ABAM without a scheduling function based on formula (3) reached 62.7%
Table 3  Effects of scheduling on throughput.

| Scheduling Function | Media Mode of Each N/I | Throughput |
|---------------------|------------------------|------------|
| –                   | 11 Mbps + 5.5 Mbps      | 6.0 Mbps   |
| √                   | 11 Mbps + 5.5 Mbps      | 7.3 Mbps   |

Table 4  Throughput for 802.11b.

| Media mode of N/I | Throughput |
|-------------------|------------|
| 11 Mbps           | 5.7 Mbps   |
| 5.5 Mbps          | 3.8 Mbps   |

Table 5  Measurement settings.

| Packet Train | Delay Monitor | Bandwidth, Delay (Link 1) | Bandwidth, Delay (Link 2) |
|--------------|---------------|-----------------------------|-----------------------------|
| Env. 1       | √             | –                           | 5.7 Mbps, 0 ms              |
|              |               |                             | 2 Mbps, 0 ms                |
| Env. 2       | √             | –                           | 5.7 Mbps, 0 ms              |
|              |               |                             | 4 Mbps, 0 ms                |
| Env. 3       | √             | –                           | 5.7 Mbps, 0 ms              |
|              |               |                             | 2 Mbps, 25 ms               |
| Env. 4       | √             | √                           | 5.7 Mbps, 0 ms              |
|              |               |                             | 2 Mbps, 25 ms               |

against the sum of the throughputs of 11 Mbps and 5.5 Mbps, listed in Table 4. However, the throughput for ABAM with a scheduling function based on formula (3) reached 76.5% against the sum of throughputs of 11 Mbps and 5.5 Mbps. This implies that the 5.5-Mbps link and 11-Mbps links mostly transmit the same amount of data using ABAM without a scheduling function. This caused the 11-Mbps link to have to wait for data to send while it could have sent more. This result shows that data transmission proportional to the available bandwidth could efficiently exploit the available bandwidth.

5.4 Packet-arrival Sequence

We measured the number-of-sequences of arriving packets exploiting ABAM with a scheduling function, which was implemented with the Packet Train method. The four experimental environments are described in Fig. 6. Table 5 lists the measurement settings. Dummynet was exploited to shape bandwidth and to generate delay.

The bandwidth in Link 2 was shaped to 2 and 4 Mbps in Environments 1 and 2. We measured the available bandwidth, but there were no delays in the environments. The bandwidth was shaped to 2 Mbps and the delay was set to 25 ms on Link 2 in Environment 3. The available bandwidth was measured but there was no delay in this environment. The bandwidth in Environment 4 was shaped to 2 Mbps and the delay was set to 25 ms on Link 2. The available bandwidth and delay were measured.

The MTU was set to 1,500 bytes for all environments.

We monitored the increase in the number-of-sequences of packets arriving at DST by sending 27-MB multimedia data. Each packet was 1,400 bytes and UDP was exploited as the transport protocol. The number of sequences, which was unique throughout the connection, was appended similarly to an SBAM header. RDTSC was exploited to measure time. The results are in Figs. 8 and 9. The vertical axis in both figures represents the number-of-sequences of packets and the horizontal axis represents the time it took the first packet sent from SRC to reach DST.

Figures 8 and 9 show that the number-of-sequences of packets arriving at DST is not out of alignment regardless of the difference between the bandwidths of two links. In addition, the increase in the number sequences for each link is proportional to the bandwidth for each link. This means that measuring the available bandwidth can efficiently be used for monitoring networks.

We next measured the queue length, which was required to re-order received packets, in SBAM at DST to clarify the efficiency of formula (2) in Environment. 3 and 4. This cor-
resides to part A in Fig. 2. We sent 27-MB multimedia data. The packets were 1,400 bytes and UDP was exploited as the transport protocol. There was a queue for each N/I. Measurement was done every 0.1 s. Figures 10 and 11 plot the results. The vertical axis represents the queue length at DST, and the horizontal axis represents the time elapsed since the first packet arrived from SRC. The queue length is represented as the number of packets. Links 1 and 2 in the figures represent the packet lengths of queues that have arrived from Links 1 and 2.

In both figures, the queue length for Link 1 is about triple that for Link 2. Furthermore, it tends to increase throughout the evaluation. For example, in Fig. 10 the queue length at 2 s is 27 packets; however, it increases to 30 packets at 3 s. Finally, it reaches 40 packets at 26 s. This implies that the algorithm needs a feedback mechanism to shorten the queue length. These figures indicate that sending the difference between the BDPs of two links does not keep the queue length short.

The increase in the number of packet sequences from the first packet to the 100th packet is plotted in Figs. 12 and 13 to clarify why the queue lengths increased. The vertical axis represents the number of packet sequences and the horizontal axis represents the time elapsed since the first packet arrived. The time for a packet to arrive from Link 2 is 0.025 s without delay being measured in scheduling algorithm. However, the time for the first packet to arrive is 0.01 s with delay being measured. This means that the timing for transmission was efficient. However, when transmission began, SRC sent 10 packets to Link 2, and thus 11 packet sequences arrived from Link 1. Therefore, the packets had to be in the queue until prior packets, from 1 to 10, arrived at Link 2, which is 0.06 s since the first packet arrived. This caused the ABAM in Fig. 11 to have a queue longer than that in Fig. 10. This may have been caused by the Dummynet, which was exploited in Link 2 to generate 25-ms delay; this means data were buffered on the Dummynet host for 25 ms. Before reaching 25 ms, SRC transmitted data on Link 1, which arrived at DST host faster than the data transmitted on Link 2. Therefore, the queue for Link 2 at DST lengthened.

These results imply that we need to devise an alternative algorithm that exploits formula (2), taking the number of packet sequences arriv-
Table 6 Ratio of an SBAM header overhead and packet existence ratio on the Internet.

| Packet size | Ratio of SBAM header overhead | Packet existence ratio on the Internet |
|-------------|------------------------------|----------------------------------------|
| 40 bytes    | 16.7%                        | 50%                                    |
| 570 bytes   | 1.4%                         | 20%                                    |
| 1,500 bytes | 0.05%                        | 15%                                    |

ing since data transmission began into account. For example, data transmission to longer-delay links begin in the 10th packet instead of in the first packet.

5.5 SBAM header overhead

As we can see from Fig. 3, a $4 + 4 = 8$ byte SBAM header is attached to each packet. The header causes a degree of overhead on the size of packets sent to the network.

McCreary and Claffy reported packet-size characteristics on the Internet\(^{16}\). They claimed about half the packets were around 40 bytes in size, 20% were around 570 bytes, and 15% were about 1,500 bytes. 40-bytes packets are mostly TCP acknowledgments. The 570-bytes packets have actual data in them, but the size is determined by the path MTU discovery mechanism. The 1,500-bytes packets also contain actual data, and are determined according to the Ethernet’s MTU size.

This tendency\(^{16}\) is based on TCP behavior. SCTP\(^6\) has recently been emerging as a transport protocol for the next generation. When this protocol becomes popular with many applications, the tendency\(^{16}\) will change; however, as far as we know, there are currently few SCTP-capable applications.

Table 6 shows the overhead ratio for an SBAM header and the existence ratio for each packet on the Internet. We can see that the overhead ratio for 40-byte packets is large. However, an SBAM header will not be attached to most 40-byte packets, since they are mostly TCP acknowledgments. The 570-bytes packets have actual data in them, but the size is determined by the path MTU discovery mechanism. The 1,500-bytes packets also contain actual data, and are determined according to the Ethernet’s MTU size.

We proposed SBAM, i.e., a Socket-level Bandwidth Aggregation Mechanism, in this paper. We implemented SBAM in a socket layer and ABAM in an application layer with several SBAM functions. SBAM and ABAM with two 802.11b N/Is achieved a factor of 1.6 increase in throughput compared with that with single N/I. The increased throughput of SBAM was 97.4% compared with that of ABAM. Many rtalloc() calls and linear searches caused degradation in throughput. We found that part A in SSF needs to monitor the number of sequences of arriving packets on DST. Part B utilizes available bandwidth effectively. We found that an 8-byte SBAM header did not impose much overhead. Although half the packets on the Internet are small 40-bytes packets, most of them have TCP acknowledgments. Therefore, SBAM headers will not be attached to these small packets. However, an additional mechanism is required to avoid packet fragmentation in the IP stack.

Future work remains to make SBAM more efficient. The most important is to implement the rest of the functions and do evaluations using all of them. Evaluating throughput with stateful transport protocols, such as TCP or SCTP, and installing more than three NICs are also required. Decreasing the receive-queue length on the destination host and increasing the throughput between two nodes are needed to achieve better performance. When there is a large difference in available bandwidth between NICs, the narrower bandwidth link should not be used for aggregation. Therefore, we need to calculate this difference accurately.

We plan to implement some functionalities and mechanisms to deploy SBAM in practice. A mechanism for assigning wireless channels automatically is needed to avoid radio interference. Since users tend to move around when using wireless networks, a seamless roaming functionality between multiple wireless regions is also required. Furthermore, dynamic NIC attachment and detachment should also be supported.

Acknowledgments This research was conducted as part of the Ubila Project supported...
by the Japanese Ministry of Internal Affairs and Communications.

References

1) IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee): Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band (2000).
2) IEEE 802.11 Standard (IEEE Computer Society LAN MAN Standards Committee): Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer in the 5 GHz band (2000).
3) Bluetooth: http://www.bluetooth.com/
4) Ultra Wideband (UWB): http://www.uwb.org/
5) IEEE 802.3ad Standard (IEEE Computer Society LAN MAN Standards Committee): Aggregation of Multiple Link Segments (2000).
6) Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and Paxson, V.: Stream Control Transmission Protocol, RFC 2960 (2000).
7) Hsieh, H.-Y., Kim, K.-H., Zhu, Y. and Sivakumar, R.: A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces, Proc. ACM MOBICOM’03 (2003).
8) Hsieh, H.-Y. and Sivakumar, R.: A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts, Proc. ACM MOBICOM’02 (2002).
9) Sakakibara, H., Moriwake, S., Saito, M. and Tokuda, H.: A Socket-level Bandwidth Aggregation Mechanism (in Japanese), Information Processing Society of Japan (IPSJ) 12th Workshop on Multimedia Communication and Distributed Systems (DPS), Vol.2004, No.15, pp.49–54 (2004).
10) Garg, S. and Kappes, M.: An experimental study of throughput for UDP and VoIP traffic in IEEE 802.11b networks, Wireless Communications and Networking, 2003 (WCNC 2003), 2003 IEEE, Vol.3, pp.1748–1753 (2003).
11) John, P.: Transmission Control Protocol (1981). RFC 793.
12) Keshav, S.: A Control-Theoretic Approach to Flow Control, Proc. conference on Communications architecture & protocols, pp.3–15 (1991).
13) Henderson, T.R. and Katz, R.H.: Transport Protocols for Internet-Compatible Satellite Networks, IEEE Journal, Vol.72, Addison-Wesley (1999).
14) Dovrolis, C., Ramanathan, P. and Moore, D.: What Do Packet Dispersion Techniques Measure?, INFOCOM, pp.905–914 (2001).
15) Rizzo, L.: Dummynet: a Simple Approach to the Evaluation of Network Protocols, ACM Computer Communication Review (1997).
16) McCreary, S. and Claffy, K.: Trends in Wide Area IP Traffic Patterns — A View from Ames Internet Exchange, ITC Specialist Seminar (2000).

Received May 19, 2006
Accepted November 2, 2006
Released February 7, 2007

Hiroshi Sakakibara is currently a Master’s student at the Graduate School of Media and Governance of Keio University. His research advisor is Professor Hideyuki Tokuda. He is interested in mobile networking, sensor networking, and ubiquitous computing. His contact information is: Keio University, Tokuda Lab., 5322 Endoh, Fujisawa, Kanagawa, 252–8520, Japan. (skk@ht.sfc.keio.ac.jp)

Masato Saito is a Ph.D. student at the Graduate School of Media and Governance of Keio University and a JSPS Research Fellow (DC2). His research advisors are Professor Hideyuki Tokuda and Professor Yoshito Tobe. His research interests are in the 3D visualization of end-host network information, and ubiquitous ad hoc networking. He is a member of ACM, IEICE, and IPSJ. His contact address is: Keio University, Tokuda Lab. 5322 Endo, Fujisawa, Kanagawa 252–8520, Japan. (masato@acm.org)
Hideyuki Tokuda received his Ph.D. in Computer Science from the University of Waterloo in 1983 and was a Senior Research Computer Scientist in the Computer Science Department at Carnegie Mellon University (CMU). He is currently the Chairman of the Graduate School of Media and Governance, and a Professor in the Faculty of Environmental Information at Keio University. His research interests include distributed real-time systems, operating systems, mobile computing, networked appliances, and ubiquitous computing systems. He is a member of ACM, IEEE, IEICE, JSSST, and IPSJ. (hxt@ht.sfc.keio.ac.jp)